# P-MoVE: Performance Monitoring and Visualization with Encoded Knowledge

Fatih Taşyaran
*Computer Science and Engineering*
*Sabancı University*
Istanbul, Turkey
fatihtasyaran@sabanciuniv.edu

Osman Yasal
*Computer Engineering*
*Koç University*
Istanbul, Turkey
oyasal22@ku.edu.tr

José A. Morgado
*INESC-ID*
*Universidade de Lisboa*
Lisbon, Portugal
jose.a.morgado@tecnico.ulisboa.pt

Aleksandar Ilic
*INESC-ID*
*Universidade de Lisboa*
Lisbon, Portugal
aleksandar.ilic@edu.ulisboa.pt

Didem Unat
*Computer Engineering*
*Koç University*
Istanbul, Turkey
dunat@ku.edu.tr

Kamer Kaya
*Computer Science and Engineering*
*Sabancı University*
Istanbul, Turkey
kaya@sabanciuniv.edu

***Abstract*—P-MoVE is a modern, open-source framework designed to monitor and visualize live and/or recorded performance data with the ultimate goal of being a digital twin for HPC systems. Leveraging a Knowledge Base (KB), built upon an HPC-specific ontology with an intuitive encoding for comprehending the performance, it rigorously manages telemetry samplers, databases, and visualization frameworks. The KB is generated through an in-depth probing of the system. It enables the configuration and monitoring of performance metric samplers, the generation of real-time visualizations, the establishment of linked-data connections, and the generation of queries for advanced analysis. Furthermore, with an *Abstraction Layer*, P-MoVE can be used for low-level profiling even on components from different vendors. It is equipped with modern profiling capabilities, including *live cache-aware roofline modeling*, crafted to provide real-time insights without impeding system performance. P-MoVE's capabilities have been demonstrated on various architectures using microbenchmarks and a common kernel, sparse-matrix vector multiplication.**

***Index Terms*—HPC, profiling, performance visualization, optimization, digital twins for HPC.**

## I. Introduction

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 performance [1]. To efficiently and effectively find the root causes of these variations, one requires a comprehensive, structured knowledge of the computational system generated via novel monitoring, profiling, and forecasting tools. Nevertheless, the diversity within HPC ecosystems brings challenges in system design, performance engineering, and optimization, necessitating innovative models and approaches to skilfully steer inside a complicated computing environment.

Various tools can systematically collect and store information from performance metric sources [2]–[5]. However, they do not create a comprehensive *knowledge base* and most do not provide a live or automated analysis framework. Hence, there is a need for tools capable of tracking every individual HPC component leveraging component-based kernel statistics and physical hardware performance counters. We propose P-MoVE, 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. Currently, the tool has modern profiling capabilities such as live/offline monitoring, visualization, and analysis e.g., *cache-aware roofline analysis*. We believe that the flexible design of P-MoVE makes it a good candidate to implement a *digital twin* of an HPC system, which can model, observe, and analyze the inherent and potential performance variability in detail. In its core, the *knowledge base* is deeply incorporated in almost every functionality. It contains the machine specification, topology, and configuration parameters of the tools/frameworks on top of an HPC-specific ontology, a must-have for digital twins. The KB also contains historical job metadata linked to the sampled performance metrics.

P-MoVE's knowledge encoding leverages the same ideas used in digital twins. While ontologies exist for various domains, such as industrial machines [6] and cities [7], there is limited work on ontologies for a computational system. Moreover, unlike existing twins collecting data from stable sources, P-MoVE 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-

This work was supported by Scientific and Technological Research Council of Turkey (TÜBİTAK), Fundação para a Ciência e a Tecnologia of Portugal (FCT) and EuroHPC Joint Undertaking through grant agreement No 220N254, UIDB/50021/2020 and grant agreement No 956213 (SparCity), respectively. Authors from Koç University were supported in part from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 949587).The numerical calculations reported in this paper were partially performed at TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources).

monitoring units (PMUs). In this work, we focus on the design and validation of P-MoVE and its potential to address the challenges posed by the complex computational system landscape. The contributions can be summarized as follows:

- We propose a modern, digital-twin-inspired design of an open-source[1] performance monitoring and visualisation framework, P-MoVE. **Live monitoring** and **cache-aware roofline analysis** are implemented on top of this to provide real-time insights with minimal interference.
- We derive an **ontology** as a guide to comprehending data center and HPC servers in an organized and intuitive manner. With this, P-MoVE constructs a **knowledge base** used to perform all the tasks on the linked performance data. To handle the architectural diversity and be as automated as possible, it leverages an **abstraction layer**.
- We have used a variety of servers to assess the practical value of P-MoVE on microbenchmarks and SpMV kernels. While doing this, we have also observed and measured the overhead incurred, the precision of measurements and the number of data points that can be collected per second.
- P-MoVE's main innovation lies in its ability to dynamically construct and maintain a detailed knowledge base that captures system behavior and interactions. This enables **comprehensive system view** and **advanced visualization** capabilities which will be useful for analyzing various components, such as memory hierarchies, cache behaviors, and computational bottlenecks, with the aim of enhancing system performance and resource management.

Overall, P-MoVE methodically paves the way in understanding and optimizing HPC systems as a comprehensive, systematically designed, open-source *digital-twin* framework. Its historical data access capability, as well as a global view with SuperDB, can be leveraged to replay or simulate various configurations to identify bottlenecks and propose potential hardware or software configurations. Although we do not touch these in this design-focused study, P-MoVE's functionalities can be extremely valuable for tasks such as predictive performance modelling on a candidate architecture, suggesting hardware upgrades, or providing novel insights.

The paper is organized as follows: Section II summarizes the related work and Section III introduces P-MoVE. Its working mechanics and additional features are given in Section IV. Section V presents the experimental results and Section VI concludes the papers and discusses future work.

## II. Background & Related Work

The related 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 [2], [3], Ganglia [8], Nagios [4], and PerfAugur [5]. E2EWatch [1] specializes in system-wide monitoring using Linux metrics and focuses on anomaly classification and detection. ClusterCockpit [9] reports performance metrics from distributed systems to InfluxDB and offers dashboards and job history queries. However, these tools have limitations, such as supporting only preselected 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 [10] and physics [11]. RDF (Resource Description Framework) is a standardized approach for organizing data as *triples*, a source node (the subject), an edge name (the predicate), and a target node (the object). To enhance this structure, RDFs incorporate extra identifiers for node descriptions and properties. JSON-LD, an RDF serialization, has unique *attributes*, differing it from the JSON format. The most common attributes are `@context`, `@id` and `type`. With these, 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 [12]. To our knowledge, no existing tool exploits this flow and generates a digital twin via linked data. Thus, there is a gap in the literature where both ends should meet.

We expect that digital twins for HPC systems will differ from others due to the abundance of sensors, where each sensor, such as a hardware register or PMU, is capable of reporting thousands of metrics through re-programming. Treating processes as unique components further adds to the heterogeneity. 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*. P-MoVE combines these components to hierarchically model an HPC system's structure, considering each individual 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 P-MoVE.

The Roofline Model [13], and its numerous variations [14]–[16], including the Cache-Aware Roofline Model (CARM) [17], have emerged as invaluable tools to evaluate the computational capabilities of contemporary processors and pinpointing potential performance limitations [18], [19]. P-MoVE 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 [20], 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 P-MoVE framework. Furthermore, this work also addresses another gap in the area of Roofline modeling in general; **real-time CARM visualization during execution**. P-

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

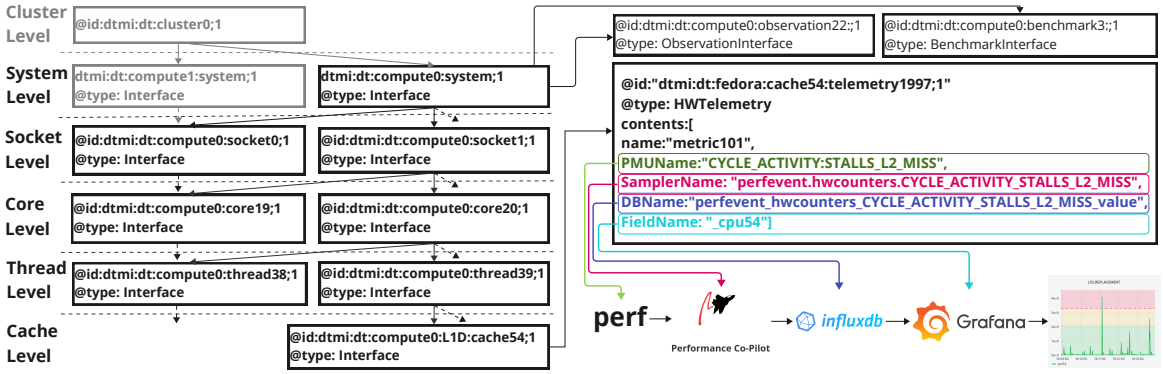


Fig. 1: Knowledge Base of P-MoVE.

MoVE 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 which is a prime example of leveraging the capabilities of P-MoVE.

## III. Modeling HPC Systems with Linked Data

P-MoVE relies on a comprehensive *Knowledge Base* (KB) and linked-data capabilities. The KB is given to each function as a parameter - it is a snapshot of every piece of information obtained from probing and previous analyses. It is dynamic and evolving to capture and link additional telemetry and metadata as they become available. This allows P-MoVE 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.

### A. 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. P-MoVE leverages Performance Co-Pilot (PCP) [21] to offer a robust and full-fledged metric collection, transport, and storage framework efficiently handling diverse hardware and performance metrics and enabling the creation of digital twins.

Here we explain how performance data is collected and incorporated into the KB. 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 [12], [22]. For this reason, P-MoVE'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.

### B. Visualization with the Knowledge Base

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) *Subtree View*, and (c) *Level 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, P-MoVE 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 the Grafana visualization tool.

- **The focus** (i.e., component) view visualizes metrics from a single component, e.g., a socket, core, thread, network, disk, or process, providing a lens on individual performance. This view can be extended to focus on the path from the root (whole system) to a unique component to investigate the root cause of anomalies 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 a cache.
- **The subtree** (i.e., (sub)system) view *zooms* into performance events, starting from an arbitrary node and extending to all connected leaf nodes, moving from a generic perspective to a more specific one, i.e., from a socket to all cores/caches. The detail level increases as the path moves from the root (subsystem) to the leaf (bottom components 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(b).
- **The level** (i.e., type) view visualizes multiple instances of the same type, such as a group of threads, disks and processes. This allows the *isolation* of a specific type and corresponds to a level in the KB tree, viewing them


(a) Focus view for an individual cache

(b) Subtree view for a node

(c) Level view for processes

(d) Level view for procs/sockets

Fig. 2: Dashboards, automatically generated by P-MoVE .

individually or in comparison. The linked-data capabilities enable the automatic visualization of performance across different machines. For instance, the level-view dashboards for different processes running SpMV on two sockets with two different orderings (none, rcm, degree, random) and on different servers (skx, icl - see experiments for details) of a matrix are given in Figs. 2(c) and 2(d).

In P-MoVE, each dashboard is only a simple JSON file (e.g., Lising 1). A dashboard can be modified by the users and saved for the next sessions. The corresponding JSON file can be shared by multiple users, etc. With a plugin, *Grafana* processes the file and handles the connections to the streaming database that stores the performance data coming from P-MoVE telemetry agents and displays them.

```
{
"id": 1
"panels":
  [{"id": 1,
    "targets":
      [{"datasource":
       {"type": "influxdb",
        "uid":  "UUkm1881"},
        "measurement": "perfevent_hwcounters_FP_ARITH_
        SCALAR_SINGLE_value",
        "params": "_cpu0"}]}]
"time":
  {"from": "now-5m",
   "to": "now"}
}
```

Listing 1: JSON for simple Grafana dashboard - From target fields datasource, uid, measurement and params are stored in STD and used to generate panel.

### C. Keeping the Knowledge Base Live

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, P-MoVE uses its *probing tool*. To comprehensively capture the system structure, including component specifications, inter/intra-relationships, and their associated performance metrics, a detailed probing is required. P-MoVE 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. The 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 [23] and `cpuid` instruction. When available, disk info is probed from `/sys/block/*/device` and `SMART` [24] utility. PMU information is collected with `libpfm4` library, which can recognize model-specific registers (and events) of virtually every x86 and ARM processor on the market. The available PMU metrics via `libpfm4` and software telemetry via PCP are filtered and mapped with the components.

In the initial KB, every component that performs computation, communication, or I/O is represented with an `Interface`. Each relationship among the components is encoded into these interfaces with a `Relationship`. The available component metrics are filtered and encoded as *SWTelemetry* and *HWTelemetry*. This enables precisely pinned executions and automated queries. To keep the KB live and continuously link the system components to performance data, P-MoVE uses `Interfaces` and attaches their instances (i.e., entries) to KB. Except for a `ProcessInterface` entry, 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 processes' dynamic nature. For performance events, P-MoVE has two other interface classes:

- `BenchmarkInterface`, and `BenchmarkResult` as a helper class, is designed to record benchmark results. P-MoVE can perform *Cache Aware Roofline Model* (CARM), *STREAM* [25] and *High Performance Conjugate Gradient* [26] (HPCG) benchmarks using the `BenchmarkInterface`. As the probing phase, P-MoVE first copies the benchmark source codes to the target system. If possible, based on the information in KB, it first compiles the benchmarks on the target system using a preferred compiler, e.g., `icc` or `gcc`. After the benchmark, P-MoVE 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 are generated to automatically retrieve data through these entries. A basic entry is shown in Listing 2. The queries automatically generated by P-MoVE to analyze the `BenchmarkEntry` in Listing 1 are given in Listing 2.

### D. Adding Compute Devices to P-MoVE

The integration of a computing device, i.e., FPGA, GPU, etc., into the KB is as easy as adding other components. Initially, an in-depth probing of the target devices is done using the available tools. In the case of NVIDIA GPUs, which is the only GPU architecture supported by the current implementation of P-MoVE, this investigation uses `nvidia-smi` to find GPUs, their models, bus and process information. `/sys/class/drm/` is used for NUMA location, and `DeviceQuery` for the HW specifications such as the number of SMs, shared memory, and cache sizes. The latest GPUs lack the capability for real-time HW telemetry reporting without source code modifications. To address this, we used `pcp-pmda-nvidia` for collecting `SWTelemetry`, essentially capturing every metric supported by NVML. Regarding `HWTelemetry`, we leveraged the approach used in benchmark executions. P-MoVE is tasked with creating a wrapper script for initiating the kernel launch and configuring `ncu` to record runtime HW performance events. Following these executions, it 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 4.

### E. Connecting P-MoVE Instances Globally

For long-term data management, P-MoVE operates a *global performance database*, SuperDB. Unlike local instances, SuperDB employs cloud instances of MongoDB and InfluxDB. With a global performance database, P-MoVE aims to accumulate 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.

```
{
  "@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-278
      e26c2-3fd3-45e4-862b-5646dc9e7aa0?time=1681499308500&
      time.window=17000"
}
```

Listing 2: 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.

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

Listing 3: Queries automatically generated by P-MoVE for the `BenchmarkInterface` entry given in Listing 1.

```
"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"
},
{
    "@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 4: An example `GPU Interface` entry which is used to monitor GPU devices on the system and profile kernel executions.

The users have the option to report their performance telemetry readings and the system's KB to SuperDB,

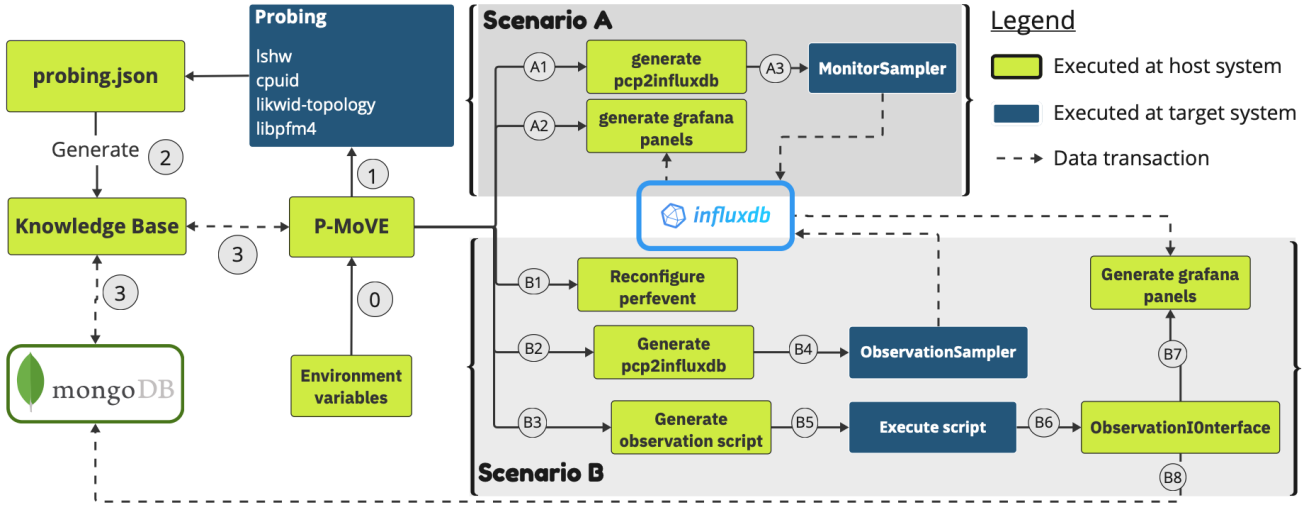


Fig. 3: Two scenarios within the P-MoVE framework.

alongside their local instances. The proposed `Observation Interface` 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 P-MoVE instance to access SuperDB, visualize performance data, and automatically generate dashboards and reports. Without P-MoVE, 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.

## IV. The Mechanics of P-MoVE

P-MoVE is designed to run on a *host* that **can be different** than the *target* system. The host runs the P-MoVE 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 3, step ⓪ reads the environment variables such as the IP addresses of InfluxDB and MongoDB instances and Grafana token to the P-MoVE daemon. In step ①, the probing module is copied to the target system to generate a JSON file containing the system information (collected from all the tools, components, and third-party tools), which, in ②, is copied back to the host to generate the KB. Once the KB is generated, it is inserted into MongoDB in step ③. Step ③ re-occurs every time KB changes or P-MoVE is restarted. When this phase is completed, the framework becomes fully functional using only this data structure.

In Figure 3, two P-MoVE scenarios are shown; the first is sampling software emitted metrics to monitor system state (Scenario A), and the other is capturing the HW performance events during kernel execution. In step (A1), using KB, P-MoVE 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.

In Scenario B, P-MoVE samples HW events from the PMUs. In this case, it focuses on the execution on the target components. It 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 P-MoVE 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.

### A. Abstraction Layer

To monitor PMU events on diverse systems hosting CPUs across various vendors and microarchitectures, P-MoVE leverages an *Abstraction Layer*. The PMUs and their events can significantly vary among different microarchitectures and from vendor to vendor. For instance, Intel has four 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. 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 I.

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

TABLE I: Intel vs. AMD PMU events: the same, similar, different, and exclusive event names for the same generic event, respectively.

To facilitate PMU event monitoring in a platform-agnostic manner, the Abstraction Layer maps generic event names to concealed HW-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 the commodity CPUs. The rest of the events are left to the user's discretion. For further flexibility, P-MoVE utilizes configuration files to establish a straightforward mapping of common events to corresponding HW 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 P-MoVE, 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 in Table II, P-MoVE's configuration mapping strategy offers versatility. Users can create mappings 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. The Abstraction Layer, which we believe is necessary for modern profiling, seamlessly generates the formulas for the user-defined events. These formulas change from vendor to vendor as well as for every architecture even when the events are the same. An illustrative use case is presented in Section V-D.

### B. Cache-aware Roofline Model in P-MoVE

*1) Model construction:* For an intuitive visualization framework, P-MoVE supports the construction of a tailored CARM model for Intel and AMD microarchitectures. It is enriched with a set of custom micro-benchmarks 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.

To assess metrics necessary for CARM roofs, such as bandwidth and peak FLOPs, we use the Time Stamp Counter (TSC) to measure the number of clock cycles, detect CPU frequency, and predefined amount of memory and compute operations contained in a specific microbenchmark. The microbenchmarks support various ISA extensions, including scalar, SSE, AVX2 and AVX512, along with multithreaded measurements. This allows for further customization of P-MoVE's CARM plot based on the prevalent ISA extension or a specific thread count utilized in the tested applications.

Thanks to KB, CARM microbenchmarks are automatically configured for a target system, taking into account cache sizes and available ISAs. To reduce the extensive benchmarking overhead of all possible thread count combinations, P-MoVE generates a subset of the most representative thread counts for the microbenchmarks. 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.

*2) Application characterization:* Besides the construction of a CARM plot for a target system, P-MoVE 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 P-MoVE 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 FLOP events of the target system, using the PMU remapping capabilities of P-MoVE. 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 FP 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 automatically retrieves the microbenchmarking results (to construct the CARM plot

of the target system) from the KB. By tightly coupling the application's live metrics with the CARM plot in the P-MoVE 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, P-MoVE 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.

## V. 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. The target systems used in the experiments are presented in Table II.

| ***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 II: Specifications of platforms used in the experiments.

### A. 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 III shows the throughput achieved with `pmdaperfevent`. Instead of sampling OS 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 observed batched zero values with high frequency. Therefore, we also count the number of zeros inserted 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 $\frac{1}{3}$ are lost on `icl`. This is due to the correlation between the loss amount and the domain size; `skx` has 88 threads, therefore there are 88 data points in each report while this number is 16 for `icl`.

| *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 III: #data points expected and observed at the host DB w.r.t. sampling freq (#samples per second) and #metrics. *(A.) Tput* inserted (actual) data points per sec.; *L+Z%* ratio of actual inserted values to the expected value.

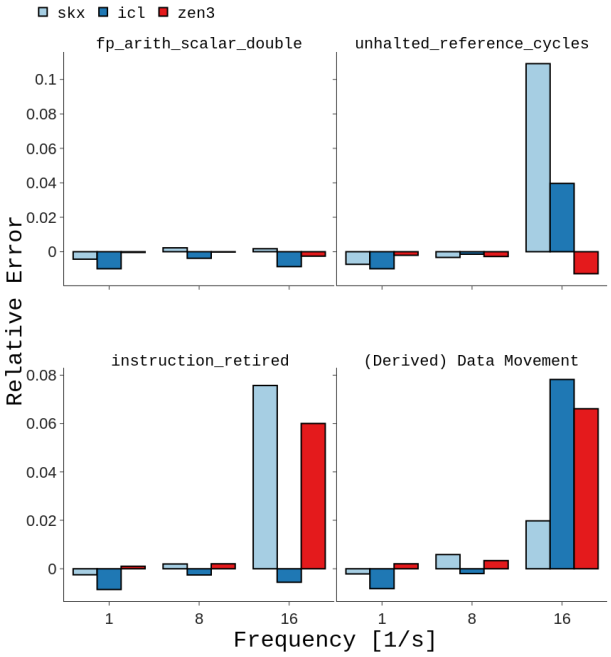


Fig. 4: Errors btw. sampled metrics and `likwid-bench` values. The $x$-axis shows #samples per second.

To verify PCP's `perf` reading accuracy while generating our performance models, we used `likwid-bench` [27], which executes a pre-determined, fixed number of instruction streams and can report ground truth for events that happened afterwards. We executed kernels `sum`, `stream`, `triad`, `peakflops`, `ddot`, `daxpy` while sampling performance, parsed the `likwid-bench` output, and compared with our readings. The relative errors acquired w.r.t. averaged kernel errors for different frequencies are reported in Fig. 4 (positive/negative values represent overcounting/undercount-

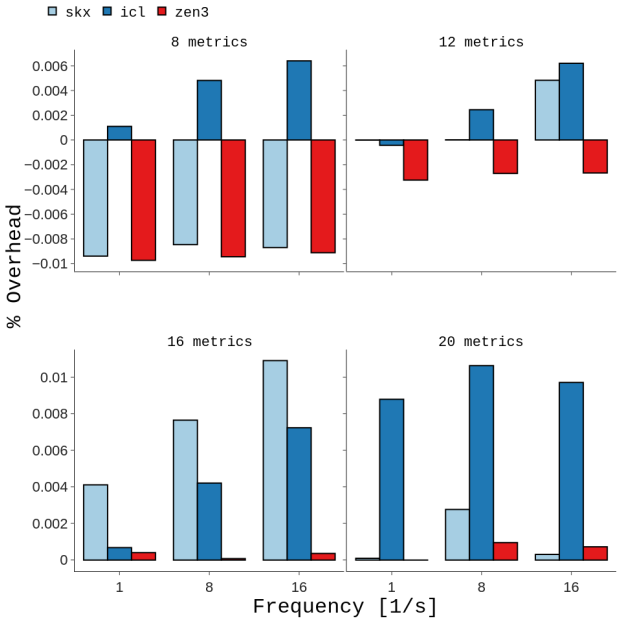


Fig. 5: Overhead caused by profiling six `likwid-bench` kernels (executions repeated 5 times, the run-times averaged). The $x$-axis shows #samples per second.

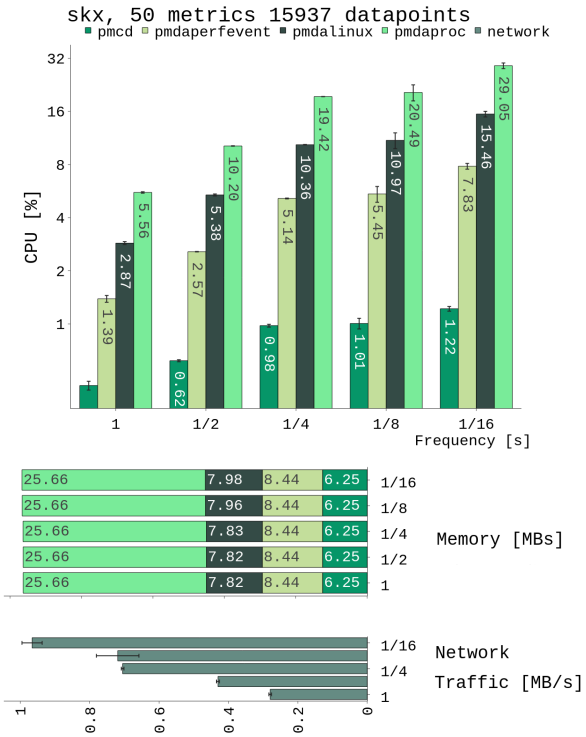


Fig. 6: System resource usage of metric shipment with kernel and PMU metrics on `skx`. The value $1/k$ on the axes implies the case with $k$ samples per second.

ing, 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 `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 [28].

### *B. Resource Usage of* P-MoVE

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; `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, P-MoVE 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. However, one case in Fig. 6 reveals that PCP does not scale perfectly for 4/8 reports per sec., with varying network traffic. This is observed in other `skx` measurements except for 10 metrics. The network and CPU under-utilization suggests that the framework may stall and fail to sample and report metrics as desired due to a lack of buffering. High-frequency sampling exacerbates this issue, leading to outdated or lost metrics in transmission, consistent with previous observations.

As CPU and network, the disk usage increases with increased sampling frequency. On a large cluster sampling with a high frequency can easily overwhelm the KB, especially in the long term and when the available storage is small. For these cases, we rely on the retention policy of InfluxDB which describes for how long the DB keeps data.

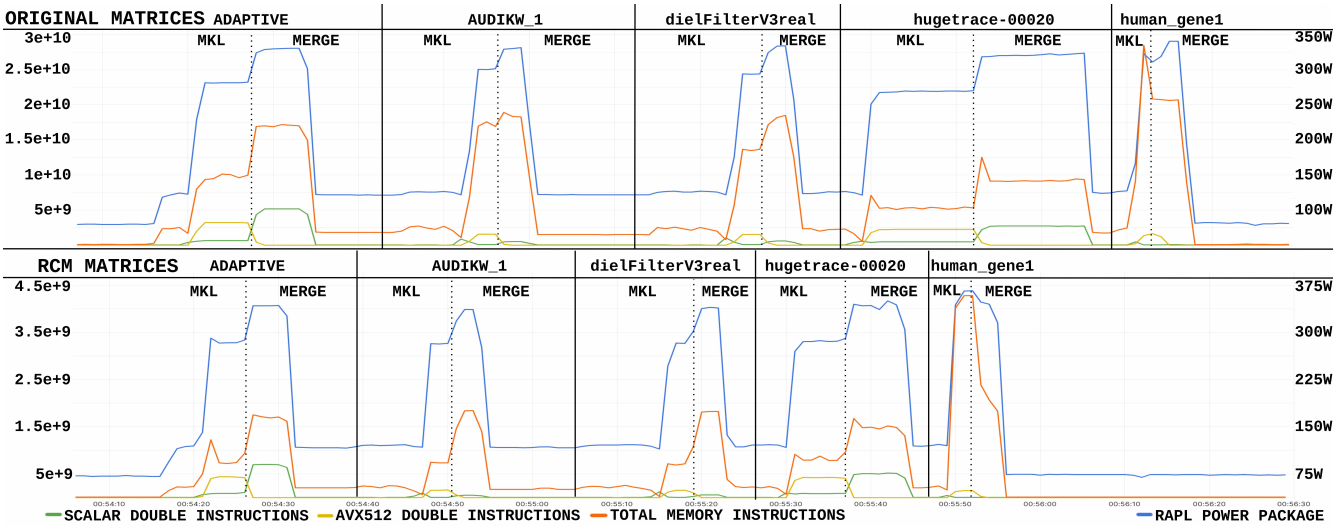


Fig. 7: Monitoring live performance events during SpMV execution on Intel CSL system

### C. Time Overhead

During HW performance event samplings, PCP-runs on the target system and performance monitoring registers are sampled. Therefore kernel run-time may be affected negatively. To measure the effect of sampling, we ran the same micro-benchmarks with/without sampling and measured the change in completion times. The overhead caused by sampling can be seen in Figure 5. 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 expected since the positive overheads are also measured at $0.01\%$. A similar negative overhead is reported by [29] even in a larger setting. However, a meaningful skew towards positive overhead is observed with increasing frequency.

### D. Monitoring Live Performance Events

To showcase the live monitoring capabilities of P-MoVE, we execute two state-of-the-art algorithms for Sparse Matrix Vector Multiplication (SpMV), i.e., Intel MKL [30] and Merge [31], on the Intel CSL system presented in Table II. We selected five sparse matrices from the SuiteSparse collection [32], as presented in Table IV, 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) [33]. For each combination of the sparse matrices, algorithms and reordering, the performance data is collected at runtime.

The results are presented in Fig. 7, with the original (top part) and RCM-reordered (bottom part) matrices. We subject each matrix to Intel MKL, followed by the Merge SpMV algorithm. For all cases, a set of PMU events were collected, including `SCALAR_DOUBLE_ INSTRUCTIONS`, `AVX512_ DOUBLE_INSTR.`, `TOTAL_MEMORY_INSTR.`, and `RAPL_POWER_PACKAGE`. Their evolution during execution is depicted in Fig. 7. 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.

| 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 IV: Sparse matrices used in the experiment.

By focusing on the evolution of collected PMU events presented in Fig. 7, one can observe that `AVX512_DP_FP` events are only manifested for Intel MKL, while `SCALAR_DP_FP` appear during the Merge algorithm. This is due to the ability of MKL SpMV to take advantage of the Intel CPU's AVX512 capabilities, while Merge SpMV only 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).

During MKL, 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. 7.

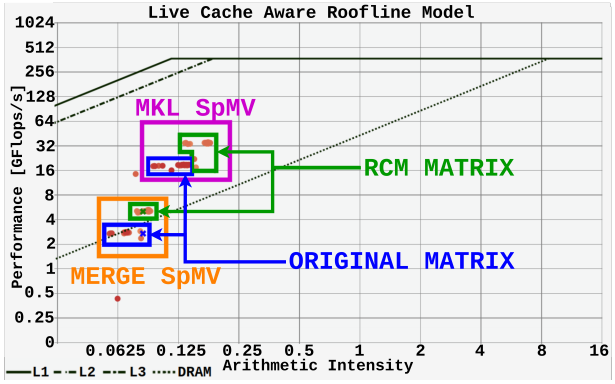


Fig. 8: Live-CARM during SpMV execution

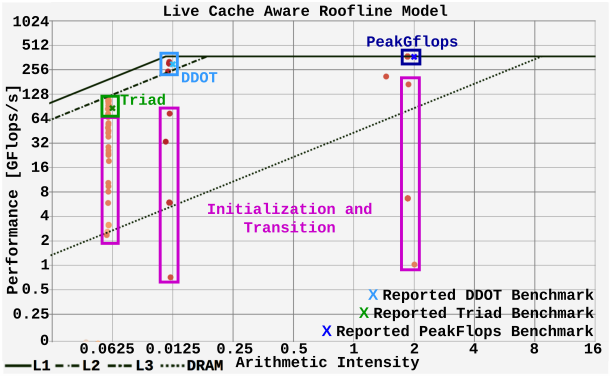


Fig. 9: Live-CARM during Likwid benchmarks execution

### E. Live-CARM feature

To showcase the live-CARM feature, we analyze the performance differences between MKL and Merge SpMV , as well as three *likwid* benchmarks on the Intel CSL system (see Tab. II).

SpMV Execution Profiling: Fig. 8 presents the live-CARM panel during both Intel MKL and Merge SpMV for `hugetrace-00020` (see Tab. IV) in its original and RCM-reordered form. The live-CARM timestamps of each execution phase are identified by the colored square that contains them, namely: *pink square* – Intel MKL; and *orange square* – Merge, while for both algorithms the *blue* and *green* squares denote the executions corresponding to the original and RCM-reordered matrix, respectively. As 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 [27] (Triad, PeakFlops, and DDOT) were considered, with corresponding live-CARM reports presented in Fig. 9.

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. 9. As can be seen, the performance of this kernel approaches the L2 roof, but it is unable to surpass 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. 9). 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. 9 (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.

## VI. Conclusion and Future Work

In this paper, we propose P-MoVE with an HPC-specific ontology, a knowledge base created on this ontology and proposed methods to parse the KB in order 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 P-MoVE, we aim to enhance the performance analysis and explainability landscape in multi-core architectures. 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 paper, enables a straightforward extension of the framework from single-node servers to clusters. Based on the proposed design in this paper, we are on the verge of developing a cluster-level P-MoVE that encapsulates meticulous performance analysis and monitoring capabilities, in conjunction with communication telemetry and job-specific metadata emitted from HPC clusters.

## References

[1] B. Aksar, B. Schwaller, V. J. L. Omar Aaziz, J. Brandt, M. Egele, and A. K. Coskun, "E2EWatch: An end-to-end anomaly diagnosis framework for production HPC systems"," in *Euro-Par 2021: Parallel Processing*. Cham: Springer International Publishing, 2021, pp. 70–85.

[2] J. M. Brandt, T. Tucker, and A. C. Gentile, "Lightweight Distributed Metric Service (LDMS): Run-time Resource Utilization Monitoring." Sandia National Lab. (SNL-CA), Livermore, CA (United States); Sandia National Lab. (SNL-NM), Albuquerque, NM (United States), Tech. Rep. SAND2013-6521C, Aug. 2013. [Online]. Available: https://www.osti.gov/biblio/1106397

[3] A. M. Agelastos, B. A. Allan, J. M. Brandt, P. Casella, j. enos, J. Fullop, A. C. Gentile, S. T. Monk, N. Naksinehaboon, J. B. Ogden, M. Rajan, M. Showerman, J. O. Stevenson, N. Taerat, and T. O. Tucker, "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications." Sandia National Labs Albuquerque, NM and Livermore, CA (United States), Tech. Rep. SAND2014-19868C, Nov. 2014. [Online]. Available: https://www.osti.gov/biblio/1315267

[4] Nagios, "Nagios," https://www.nagios.org/, 2022, accessed: 2022-12-12.

[5] S. Roy, A. C. König, I. Dvorkin, and M. Kumar, "PerfAugur: Robust diagnostics for performance anomalies in cloud services," in *2015 IEEE 31st International Conference on Data Engineering*, 2015, pp. 1167–1178.

[6] C. Steinmetz, A. Rettberg, F. G. C. Ribeiro, G. Schroeder, and C. E. Pereira, "Internet of things ontology for digital twin in cyber physical systems," in *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*, 2018, pp. 154–159.

[7] T. Deng, K. Zhang, and Z.-J. M. Shen, "A systematic review of a digital twin city: A new pattern of urban governance toward smart cities," *Journal of Management Science and Engineering*, vol. 6, no. 2, pp. 125–134, 2021.

[8] Ganglia, "Monitoring system," 2022. [Online]. Available: http://ganglia.sourceforge.net/

[9] "Cluster cockpit," https://www.clustercockpit.org/, acc. on 30 Sep 2023.

[10] J. Xin, C. Afrasiabi, S. Lelong, J. Adesara, G. Tsueng, A. I. Su, and C. Wu, "Cross-linking BioThings APIs through JSON-LD to facilitate knowledge exploration," *BMC Bioinformatics*, vol. 19, 02 2018.

[11] X. Chen, S. Dallmeier-Tiessen, A. Dani, R. Dasler, J. D. Fernández, P. Fokianos, P. Herterich, and T. Šimko, "Cern analysis preservation: A novel digital library service to enable reusable and reproducible research," in *Research and Advanced Technology for Digital Libraries*. Cham: Springer International Publishing, 2016, pp. 347–356.

[12] M. Friedemann, K. Wenzel, and A. Singer, "Linked data architecture for assistance and traceability in smart manufacturing," *MATEC Web of Conferences*, vol. 304, p. 04006, 01 2019.

[13] N. Ding and S. Williams, "An instruction roofline model for GPUs," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2019, pp. 7–18.

[14] T. Koskela, Z. Matveev, C. Yang, and A. e. a. Adedoyin, "A novel multi-level integrated roofline model approach for performance characterization," in *33rd Int. Conf, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, 33*. Springer, 2018, pp. 226–245.

[15] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, "A roofline model of energy," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 661–672.

[16] A. Ilic, F. Pratas, and L. Sousa, "Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 52–58, 2016.

[17] ——, "Cache-aware roofline model: Upgrading the loft," *IEEE Computer Architecture Letters*, vol. 13, no. 1, pp. 21–24, 2013.

[18] D. Doerfler, J. Deslippe, S. Williams, L. Oliker, B. Cook, T. Kurth, M. Lobet, T. Malas, J.-L. Vay, and H. Vincenti, "Applying the roofline performance model to the Intel Xeon Phi Knights Landing processor," in *ISC High Performance 2016 International Workshops, Frankfurt, Germany, June 19–23, 2016, Revised Selected Papers 31*. Springer, 2016, pp. 339–353.

[19] D. Unat, C. Chan, W. Zhang, S. Williams, J. Bachan, J. Bell, and J. Shalf, "Exasat: An exascale co-design tool for performance modeling," *The Int. J. of High Perf. Computing Applications*, vol. 29, no. 2, pp. 209–232, 2015.

[20] D. Marques, A. Ilic, Z. A. Matveev, and L. Sousa, "Application-driven cache-aware roofline model," *Future Generation Computer Systems*, vol. 107, pp. 257–273, 2020.

[21] "Performance co-pilot," https://pcp.io/, accessed on 30 Sep 2023.

[22] K. Milenković, S. Mayer, K. Diwold, and J. Zehetner, "Enabling knowledge management in complex industrial processes using semantic web technology," in *Proceedings of the 2019 International Conference on Theory and Applications in the Knowledge Economy*, Jul. 2019.

[23] T. Röhl, J. Eitzinger, G. Hager, and G. Wellein, "Likwid monitoring stack: A flexible framework enabling job specific performance monitoring for the masses," in *2017 IEEE Int. Conf. on Cluster Comp.*, 2017, pp. 781–784.

[24] T. S. Team. Smartmontools. Accessed on 5th October 2023. [Online]. Available: https://www.smartmontools.org/

[25] J. McCalpin, "Memory bandwidth and machine balance in high performance computers," *IEEE Tech. Comm. on Comp. Arch. Newsletter*, pp. 19–25, 1995.

[26] J. Dongarra and M. A. Heroux, "Toward a new metric for ranking HPC systems," *Sandia Report, SAND2013-4744*, vol. 312, p. 150, 2013.

[27] T. Röhl, J. Treibig, G. Hager, and G. Wellein, "Overhead analysis of performance counter measurements," in *2014 43rd International Conference on Parallel Processing Workshops*, 2014, pp. 176–185.

[28] V. Weaver, D. Terpstra, and S. Moore, "Non-determinism and overcount on modern hardware performance counter implementations," in *Prof. of the IEEE Int. Symp. on Perf. Analysis of Systems and Software*, 04 2013, pp. 215–224.

[29] A. Nowak and G. Bitzes, "The overhead of profiling using PMU hardware counters," Jul. 2014. [Online]. Available: https://doi.org/10.5281/zenodo.10800

[30] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, *Intel Math Kernel Library*. Cham: Springer International Publishing, 2014, pp. 167–188. [Online]. Available: https://doi.org/10.1007/978-3-319-06486-4_7

[31] D. Merrill and M. Garland, "Merge-based SpMV using the CSR storage format," *Acm Sigplan Notices*, vol. 51, no. 8, pp. 1–2, 2016.

[32] T. Davis. Sparse matrix collection. Accessed on 5th October 2023. [Online]. Available: https://sparse.tamu.edu/

[33] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," in *Proceedings of the 1969 24th national conference*, 1969, pp. 157–172.