Visualization Tools

Deliverable No: D4.3
Deliverable Title: Visualization Tools
Deliverable Publish Date: 30 June 2023

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 objective of this deliverable is to provide a technical overview of the research activities carried out

1.2 DEVIATIONS AND COUNTER MEASURES

1.3 RESOURCES

The following is the list of websites that host software and data repositories that have so far been developed for the SparCity framework. It is expected that these websites will later receive a more coherent organization.

- Deliverables that are public are available in the project website

  http://sparcity.eu

- Source code developed in this project is available at the project github repository
  https://github.com/sparcityeu
- A repository of sparse problem instances
  https://datasets.simula.no/sparcity

2 SparCity Visualization Tools

Digital Twins are used to provide a digital projection of cyber-physical systems via describing data-emitting sources and relationships between physical systems components. Collections of methodologies that are used to describe these pieces of information are called ontologies. Some of the well-known ontologies are SOSA (Sensor, Observation, Sample, Actuator),[1] which is used to describe industrial pipelines, and FOAF (Friend of a Friend), which is used to describe relations among people. Moreover, there are several vocabularies used to describe digital twins, such as RDF (Resource Description Framework) and OWL (Web Ontology Language). Ontologies using these vocabularies allow static information to be located and queried using web interfaces via SPARQL endpoints. These frameworks are widely used to represent web-based interactions. Digital twins of HPC systems or computers, in general, have to differ from other physical entity twins due to several reasons; (1) there are (usually) much more sensors for each subdomain of the physical system, (2) each sensor can report up to thousands of metrics, and metrics from the same sensor can vary even for the same system, and (3) when counting processes as a component of the system, even components change rapidly.

Linked data is used to generate a network of discrete and distinct entities in order to enable queries and complex analysis over different domains of data sources and interoperability. Linked data is widely adopted in web technologies and used in different science branches for knowledge management, such as biology[2] and physics.[3] RDF is a standard for data exchange graph data on the web. In the context of RDF data, an edge is referred to as a triple and consists of a source node (called a subject), an edge name (called a predicate), and a target node (called an object). An RDF graph is defined as a set of RDF triples that follow this structure. On top of this structure, RDFs have identifiers, called IRIs to unambiguously identify and properties to describe the nodes. JSON-LD is a notation used to express RDF data using JSON syntax. This means a JSON-LD document is both a JSON document and an RDF document. JSON-LD has ”ld attributes”, which separates JSON-LD from ordinary JSON. Most relevant of these attributes are; `@context`, `@id` and `type`. With these attributes, we know what a JSON-LD dictionary describes, what kind of datatypes it includes, and how to parse and process them. This is an important aspect since, in turn, it allows create big and interconnected systems from building blocks. Albeit very useful in creating knowledge graphs, these ontologies are designed and used to keep static metadata. New triples need to be injected into the graph to add new data points, making these types of ontologies impractical with using time-series data without modification.[4]

In SuperTwin, linked data paradigm employing JSON-LD is used to encode pointers and parameters of collected time-series data into SuperTwin Description and generate dashboard for different components or runs of systems, provide instant cross-comparison capabilities.

[1]Janowicz. “SOSA: A lightweight ontology for sensors, observations, samples, and actuators”. *Journal of Web Semantics* 56 (2019), pp. 1–10. issn: 1570-8268. doi: https://doi.org/10.1016/j.websem.2018.06.003. url: https://www.sciencedirect.com/science/article/pii/S1570826818300295.

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

[3]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.

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

2.1 VISUALIZATION STRUCTURES ON THE BACKEND

Digital Twin Description Language is developed by Microsoft and is a derivation of RDF. DTDL is made up of six metamodel classes that describe the context of digital twin components. These classes are; `Interfaces`, `Telemetry`, `Properties`, `Commands`, `Relationships` and `Data Types`. In DTDL, every `Interface` is a digital twin on its own, with its contents describing its `Properties`, `Telemetry`, and `Relationships`. When combined, these enable to capture of the hierarchical structure of a computer and model of every single component (e.g., CPU, GPU, memory subsystem, etc.) as a separate digital twin entity. The idea that every interface is considered a digital twin on its own is heavily exploited in SuperTwin. In SuperTwin, STD both; captures a semantic description of the target system and enables a linked time-series structure similar to the framework proposed in[5] but with far richer metadata and contemporary metadata instantiations of events. For example, individual components, observations, and processes also have their digital twin descriptions with linked time-series data. This enables a fine-grain analysis of the behavior of applications to run on different systems with different software and hardware. For example, an L1 cache, a network interface, or a process could be isolated from the system, analyzed separately, or compared with its equivalent on a different system.

DTDL have a recursive structure that allows components (interfaces in the context of twin description) to be other components' subcomponents which is crucial for describing a cyber-physical system. However, since DTDL is designed with IOT systems in mind, its descriptions are more physical (emphasising spatial relations, for example a switch is on the wall) than cyber-physical and are meant to be static. To this end, DTDL is modified, and new classes and properties are added to describe high-performance computing systems and create linked time-series data. The updates made on existing DTDL to acquire part of STD ontology can be seen in Table 1.

[5]Friedemann, "Linked Data Architecture for Assistance and Traceability in Smart Manufacturing".

Property	*Description*
@type	**Interface**
@id	Unique identifier within digital twin for interface
contents	a set of Interface, Process Interface, ObservationInterface, SWTelemetry, HWTelemetry, Benchmark, Properties, Relationships
displayName	Name to be displayed when instantiated
dashboard	dashboard url, optional
@type	**SWTelemetry**
@id	Unique identifier within digital twin for this telemetry instance
name	index in telemetries
instance	instance name of reported component to be a parameter in queries
samplerName	name of the metric to be referred to during sampler configuration
DBName	name of the metric to be used in the generation of queries
@type	**HWTelemetry**
@id	Unique identifier within digital twin for this telemetry instance
name	index in telemetries
instance	instance name of the reported component to be a parameter in queries
samplerName	name of the metric to be referred to during sampler configuration
DBName	name of the metric to be used in the generation of queries
PMUName	name of the metric as reported by libpfm4. To be used as parameter in perf event configuration
@type	**BenchmarkInterface**
@id	Unique identifier within digital twin for interface
contents	BenchmarkResult
displayName	Name of the benchmark to be displayed when instantiated
@type	**BenchmarkResult**
@id	Unique identifier within digital twin for this telemetry instance
field	name of field for subkernels, optional
no_threads	number of threads used
involved_threads	involved thread indexes to be used in queries
modifier	modifications in pinning strategy or compilation
result	result of benchmark
unit	unit of benchmark result
sampled_sw_metrics	sampled software metrics during execution, to be used in queries, optional
sampled_hw_metrics	sampled hardware metrics during execution, to be used in queries, optional
dashboard	dashboard url of observed metrics, optional
@type	**ObservationInterface**
@id	Unique identifier within digital twin for interface
displayName	Name to be displayed when instantiated
time	duration of observation
command	executed command
tag	tag of affiliated data in the database
no_threads	number of threads used
involved_threads	involved thread indexes to be used in queries
sampled_sw_metrics	sampled software metrics during execution, to be used in queries
sampled_hw_metrics	sampled hardware metrics during execution, to be used in queries
modifier	any modification made to the environment, optional
dashboard	dashboard url of observed metrics, optional

Table 1 *New metamodel classes added to DTDL to build STD. There is also a model named* `ProcessInterface`*, which is in shape identical to* `Interface`*; however, its content fields are re-assigned every time the corresponding process's pid is changed.*

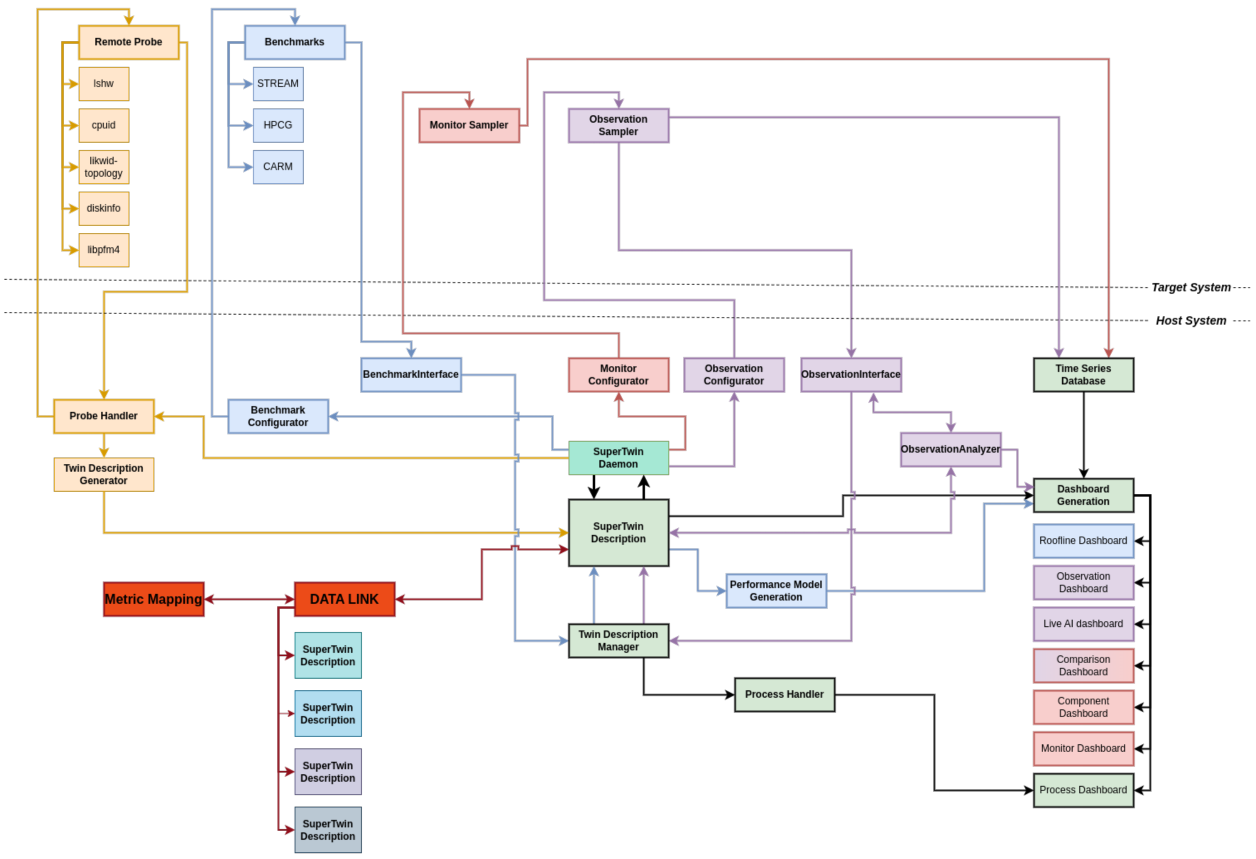


Figure 1 *Current status of SuperTwin modules.*

The current status of the SuperTwin modules can be seen in Figure 1. Different colors in the figure highlight different pipelines of operations. A SuperTwin Description (STD) is generated after a probing of the target system. During the probing, information from the environment and other collaboratively used frameworks such as Grafana, InfluxDB and Performance Co-Pilot is fused with components of the target system. This STD includes every component of the target system and the parameters for the time-series data they generate. A twin description manager module appends new information collected from the system, re-instantiates parameters for rapidly changing components, such as executed benchmarks and processes and keep STD up to date with the corresponding system. Data collection framework is configured and launched by SuperTwin daemon using STD and necessary information to query this information is appended to STD as well. After data collection starts, dashboard generation module is able to generate dashboards automatically and on demand. These dashboards could visualize system features, live and historical data and combinations of them. Since every STD uses the same syntax, a data-link between different STDs that belong to different systems make their data available to each other for comparison and further analysis. To be able to have a data-link, a user needs to either possess STD of both systems and the data that is extracted via ObservationInterfaces or password for the specific machine's data from a remote InfluxDB instance.

Listing 1: A (very small) subset of SuperTwin Description.

```
{
 "twin_description": {
  "dtmi:dt:dolap:system:S1;1": {
   "@type": "Interface",
   "@id": "dtmi:dt:dolap:system:S1;1",
   "@context": "dtmi:dtdl:context;2",
   "contents": [
    {
     "@id": "dtmi:dt:dolap:os:O1;1",
     "@type": "Property",
     "name": "os",
     "description": "Ubuntu 22.04.1 LTS"
    },
    {
     "@id": "dtmi:dt:dolap:system:telemetry41373;1",
     "@type": "SWTelemetry",
     "name": "metric0",
     "displayName": "value",
     "SamplerName": "kernel.all.load",
     "DBName": "kernel_all_load"
    }
   ]
  },
  "dtmi:dt:dolap:socket0;1": {
   "@type": "Interface",
   "@id": "dtmi:dt:dolap:socket0;1",
   "@context": "dtmi:dtdl:context;2",
   "contents": [
    {
     "@id": "dtmi:dt:dolap:socket0:property0;1",
     "@type": "Property",
     "name": "model",
     "description": "Intel(R) Xeon(R) Gold 6152 CPU @ 2.10GHz"
    },
    {
     "@id": "dtmi:dt:dolap:socket0:telemetry41431;1",
     "@type": "SWTelemetry",
     "name": "metric11",
     "displayName": "_node0",
     "SamplerName": "kernel.pernode.cpu.irq.soft",
     "DBName": "kernel_pernode_cpu_irq_soft"
    }
   ]
  },
  "dtmi:dt:dolap:thread0;1": {
   "@type": "Interface",
   "@id": "dtmi:dt:dolap:thread0;1",
   "@context": "dtmi:dtdl:context;2",
   "contents": [
    {
     "@id": "dtmi:dt:dolap:thread0:telemetry44038;1",
     "@type": "SWTelemetry",
     "name": "metric0",
     "displayName": "_cpu0",
     "SamplerName": "kernel.percpu.interrupts",
     "DBName": "kernel_percpu_interrupts"
    },
    {
     "@id": "dtmi:dt:dolap:thread0:telemetry44184;1",
     "@type": "HWTelemetry",
     "name": "metric146",
     "displayName": "_cpu0",
     "SamplerName": "perfevent.hwcounters.L1D_PEND_MISS_EDGE",
     "DBName": "perfevent_hwcounters_L1D_PEND_MISS_EDGE_value",
     "PMUName": "L1D_PEND_MISS:EDGE"
    }
   ]
  }
 }
}
```

A small subset of SuperTwin Description can be found in Listing 1. The "@id" fields that are tagged with numbers in parentheses are entries that are used in different visualization scenarios for their corresponding owner components. From those entries; those designated with the tag (1) are used to generate info cards which display their description. Tags tagged with (2) are used to generate timeseries or stat panels. Note that their "displayName" fields changing w.r.to their owner component. This field is used to both select a specific component from many from database and display their name is panels. Entry tagged with (3) is a PMU event. This event have an additional PMUName field which is used to configure perfevent sampler before sampling take place.

2.2 PROFILING CAPABILITIES

As previously mentioned in our deliverables, we selected Performance Co-Pilot (PCP) for telemetry collection, a software developed by Red Hat and widely utilized by prominent organizations such as IBM, Netflix, and CERN.

PCP adopts a modular structure that incorporates various Performance Metrics Domain Agents (PMDAs) to gather metrics from different sources. These metrics encompass system-level measurements that contribute to modeling system performance and diagnosing anomalies. They originate from various components within a system, including the kernel, memory, disks, networks, and similar entities. Although system metrics do not directly pinpoint the causes of application performance issues, they provide insights into the system's state during execution and can uncover system-related anomalies such as resource contention, thermal throttling, memory leaks, or suboptimal affinity. Consequently, these metrics need to be continuously collected at a relatively low frequency. Additionally, there are hardware performance events reported by Performance Monitoring Units (PMUs), which directly impact application performance. These events include cache misses, floating-point operations, branch misses, executed instructions, and more. Collecting these metrics requires a higher frequency, particularly during kernel execution.

PCP's PMDAs operate on a pull-only basis, which means they report metrics only when requested by a sampler. SuperTwin leverages this characteristic of PMDAs. Monitor samplers are configured and launched to remain active at all times after the installation of SuperTwin. Conversely, observation samplers are configured and launched just prior to the execution of a task, solely collecting metrics during that specific execution period. Since SuperTwin encodes all the necessary parameters for the environment and every potential metric that the system can report within its SuperTwin Description, it has the capability to configure and launch an arbitrary number of samplers upon request via the SuperTwin API.

The distinction between monitor and sampler is not strict. For example we prefer to put RAPL energy metrics together with monitor metrics. Moreover, when PMU metrics monitored constantly, they can be used to generate real time performance models. To this end, we put component metrics for CARM under constant sampling and was able to observe Arithmetic Intensity and Bandwith of each core in real-time, within SuperTwin dashboards.

2.2.1 OBSERVATION INTERFACE

To be able to recall the collected data and generate visualizations for a specific execution, or a specific interval of time, an ObservationInterface is generated and appended into SuperTwin. This ObservationInterface acts as a formula to generate necessary queries. During observations, collected telemetry is inserted into the database with their corresponding observation id as their tag, therefore could be queried easily using this tag. To query monitor metrics that collected from the system during the execution, first and last timestamp from a observation is used.

Listing 2: An example ObservationInterface entry

```
{
"@type": "ObservationInterface",
"@id": "278e26c2-3fd3-45e4-862b-5646dc9e7aa0",
"displayName": "rcm_rma10_mt",
"time": 48.667,
"command": "./spmv -f rma_10.mtx -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/KG3V5WEVz/pmus-1b39cf64-5551-49f9-a92c-777872?time=1681500242500&time.window=21000$ "
}
```

Listing 3: Example queries that is generated using ObservationInterface in Listing 2

```
SELECT "_cpu0", "_cpu1", "_cpu22", "_cpu23" FROM "kernel_percpu_cpu_idle" WHERE tag="
    ↪ 278e26c2-3fd3-45e4-862b-5646dc9e7aa0" AND time >= 1687649768499ms and time <=
    ↪ 1687650049919ms
SELECT "_node0", "_node1" FROM "mem_numa_alloc_hit" WHERE tag="278e26c2-3fd3-45e4-862b
    ↪ -5646dc9e7aa0" AND time >= 1687649768499ms and time <= 1687650049919ms
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"
```

2.3 MODELLING CAPABILITIES

Our primary focus for benchmarks and performance modeling is CARM (Cache Aware Roofline Model), while STREAM and HPCG benchmarks are also included to showcase the versatility of our proposed approach, as they are widely used for simulating HPC workloads. The integration of benchmarks into SuperTwin is achieved by incorporating their source code along with the SuperTwin framework. However, these source codes are not simply scripted; they are managed as modules within SuperTwin. During the probing phase, the source codes are copied to the target machine and compiled there, optimizing them for maximum vector size, available compiler (GCC vs. Intel), or architecture-specific features. Subsequently, a script is generated that consists of a predefined set of threads and possible placements, which is then executed on the target host. By default, this script generates a set of threads ranging from 1 to the number of threads as powers of 2, placed either on a single NUMA node or distributed evenly across NUMA nodes. Additionally, threads per core are pinned to specific cores, and threads per socket are pinned to particular sockets. The results obtained from these benchmarks are parsed and encoded into the SuperTwin Description using a BenchmarkInterface. The BenchmarkInterface is essentially an extension of the ObservationInterface, enriched with metadata relevant to benchmarks. These benchmark results can be accessed at any time for the purpose of reconstructing performance models and evaluating executed applications.

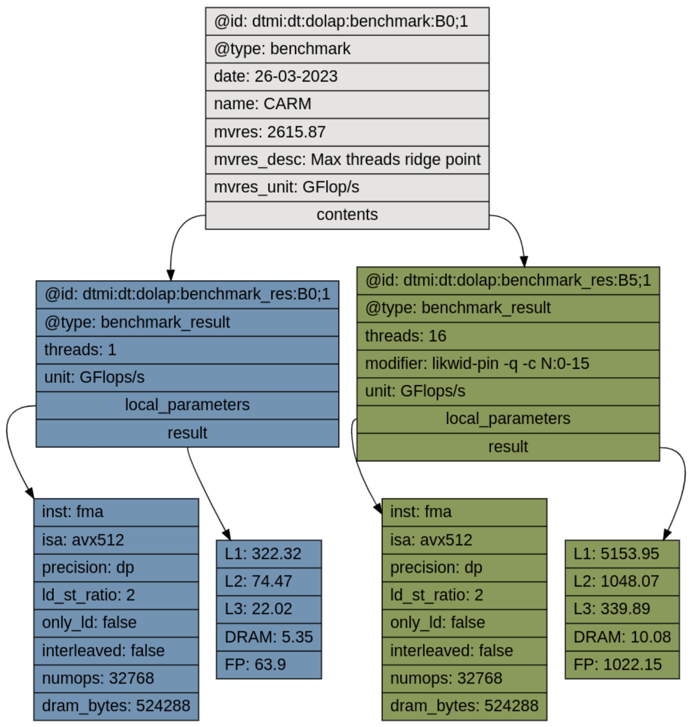


Figure 2 *An example BenchmarkInterface that encodes instances of CARM benchmark with different settings.*

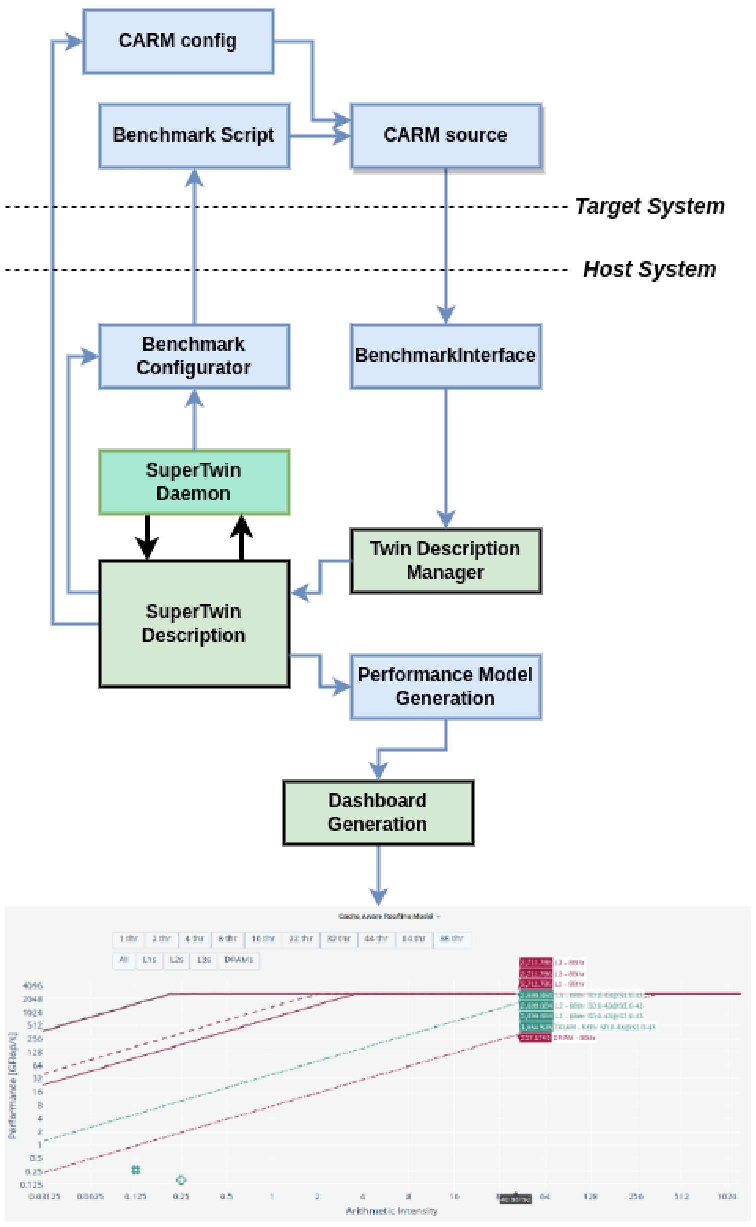


Figure 3 *Execution steps and involved modules of SuperTwin to generate a performance model in SuperTwin dashboard*

In addition, monitoring capabilities are provided for benchmarks, allowing for performance event comparisons and performance analysis across different machines. This functionality enables the generation of metrics such as GFlops per watt for various thread pinning or NUMA placement configurations on-the-fly.

2.4 VISUALIZATION FRAMEWORK

2.4.1 GRAFANA

Grafana is an open-source visualization tool that provides dynamic dashboards, ad-hoc queries, and alerting functions on time-series data. Since it's initial release, it quickly become the industry standard and reached 10M+ global users. Due to its massive user-base, Grafana supports every popular database and provides a wide variety of visualization methods. Grafana dashboards are serialized JSON files that could be templated, uploaded, and altered via web API. Since the dashboards are actually only JSON files, they are very easy to manipulate and generate for numerous metrics, and also easy to interact with SuperTwin Description (STD).

2.4.2 Generating Grafana panels using SuperTwin

Grafana dashboards being serialized JSON objects is extensively utilized in SuperTwin. Since all the necessary parameters for query generation are already encoded in STD the process of generating a dashboard panel is simply becomes transitioning from an arbitrary STD interface to a Grafana object. There are many different panel types available with Grafana. From those; stat, timeseries and gauge panels are tailored for their explanatory and visual aspects, then matched with types of metrics that is collected and used in SuperTwin dashboards. On top of native Grafana panels, Plotly charts also could be displayed within Grafana dashboards. In SuperTwin, system features and benchmark results, or any static data is visualized using Plotly graph object notation, which is also described in JSON, therefore transition of data and functions are very similar to generating Grafana panels. Categorical values, such as benchmark results are converted into Plotly traces and used in panels that provide interactive selection and visualization alongside native Grafana panels. Informative values, such as system features, or change in time w.r. to other executions are visualized using plotly indicators.

Dashboards can be dynamically generated for each interface and it's contents that includes performance metrics or static information. This flexibility allows for the creation of dashboards on-the-fly, incorporating the specific metrics and configurations relevant to each component. Moreover, employing ObservationInterface and linked-data paradigm; dashboards for specific observations and cross-comparisons between different runs on the same machine or different machines could be generated.

Listing 4: JSON for simple Grafana panel - From target fields datasource, uid, measurement and params are stored in STD and used to generate panel.

```
{
"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"}
}
```

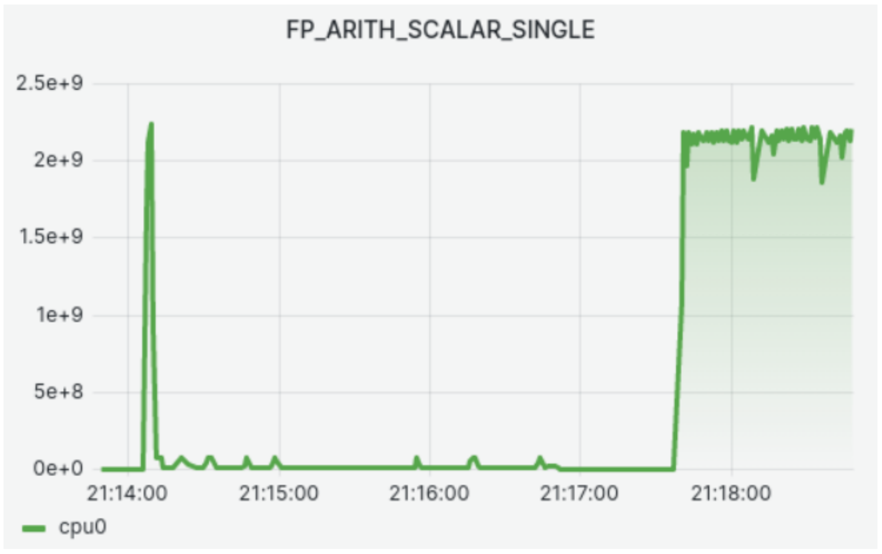
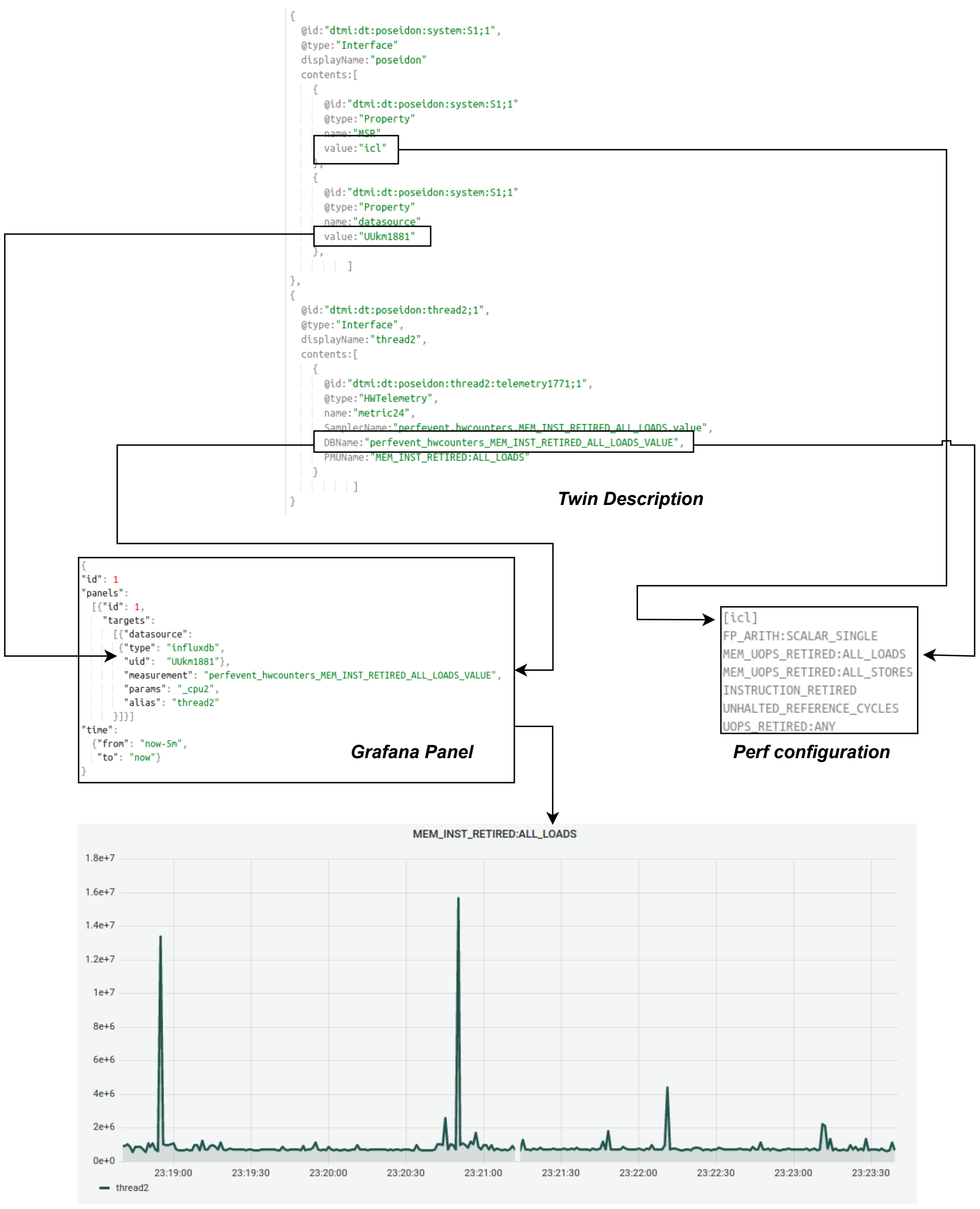


Figure 4 *Grafana panel generated when JSON from Listing 4 is serialized. SuperTwin dashboards are generated as collections of these panels, together with on-the-fly analysis panels.*



```
{
  @id:"dtmi:dt:poseidon:system:S1;1",
  @type:"Interface"
  displayName:"poseidon"
  contents:[
    {
      @id:"dtmi:dt:poseidon:system:S1;1"
      @type:"Property"
      name:"MSR"
      value:"icl"
    },
    {
      @id:"dtmi:dt:poseidon:system:S1;1"
      @type:"Property"
      name:"datasource"
      value:"UUkm1881"
    },
          ]
},
{
  @id:"dtmi:dt:poseidon:thread2;1",
  @type:"Interface",
  displayName:"thread2",
  contents:[
    {
      @id:"dtmi:dt:poseidon:thread2:telemetry1771;1",
      @type:"HWTelemetry",
      name:"metric24",
      SamplerName:"perfevent.hwcounters.MEM_INST_RETIRED_ALL_LOADS.value",
      DBName:"perfevent_hwcounters_MEM_INST_RETIRED_ALL_LOADS_VALUE",
      PMUName:"MEM_INST_RETIRED:ALL_LOADS"
    }
           ]
}
```

Twin Description

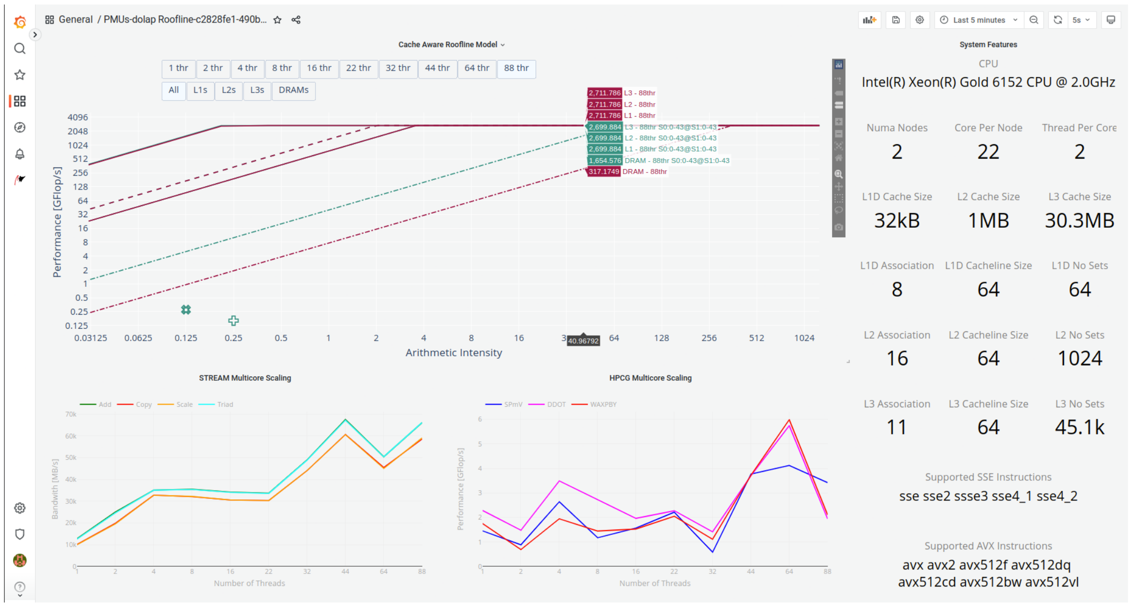
```
{
"id": 1
"panels":
  [{"id": 1,
    "targets":
      [{"datasource":
       {"type": "influxdb",
        "uid":  "UUkm1881"},
        "measurement": "perfevent_hwcounters_MEM_INST_RETIRED_ALL_LOADS_VALUE",
        "params": "_cpu2",
        "alias": "thread2"
      }]}]
"time":
  {"from": "now-5m",
   "to": "now"}
}
```

Grafana Panel

```
[icl]
FP_ARITH:SCALAR_SINGLE
MEM_UOPS_RETIRED:ALL_LOADS
MEM_UOPS_RETIRED:ALL_STORES
INSTRUCTION_RETIRED
UNHALTED_REFERENCE_CYCLES
UOPS_RETIRED:ANY
```

Perf configuration

Figure 5 *Usage of SuperTwin Description for metric collection and visualization purposes. Using the encoded parameters, perfevent collection agent is configured, then a panel for the metric and it's queries are generated.*



CPU
Intel(R) Xeon(R) Gold 6152 CPU @ 2.0GHz

Numa Nodes	Core Per Node	Thread Per Core
2	22	2
L1D Cache Size	L2 Cache Size	L3 Cache Size
32kB	1MB	30.3MB
L1D Association	L1D Cacheline Size	L1D No Sets
8	64	64
L2 Association	L2 Cacheline Size	L2 No Sets
16	64	1024
L3 Association	L3 Cacheline Size	L3 No Sets
11	64	45.1k

Supported SSE Instructions
sse sse2 ssse3 sse4_1 sse4_2

Supported AVX Instructions
avx avx2 avx512f avx512dq avx512cd avx512bw avx512vl

Figure 6 *A roofline dashboard with combined results of CARM, STREAM, HPCG and system specifications. As mentioned, benchmark results are stored as BenchmarkInterface in the twin description. Note that, because the metrics required to calculate Arithmetic Intensity of an execution is collected during the execution of benchmarks, STREAM and HPCG benchmarks are marked on the roofline. Storing benchmark results in STD allows re-generation of the roofline model and mark the executed application if required metrics are collected. Instability in STREAM and HPCG benchmark's results are due to incomplete NUMA placement module.*

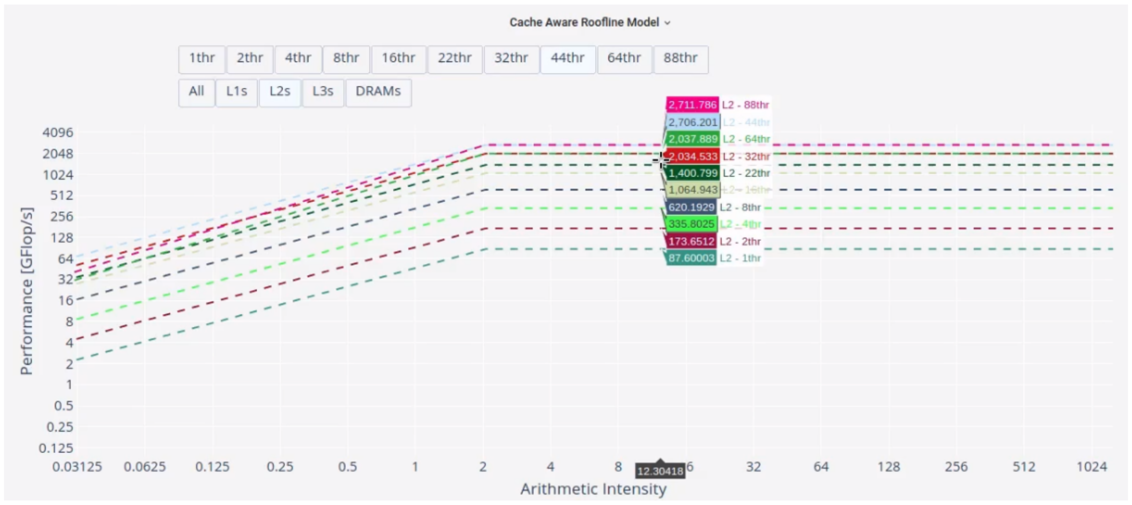


Figure 7 *Roofline dashboard model generated with plotly is interactive and allows selecting data. In this figure, peak performance of L2 caches with different multi-threading settings is selected.*

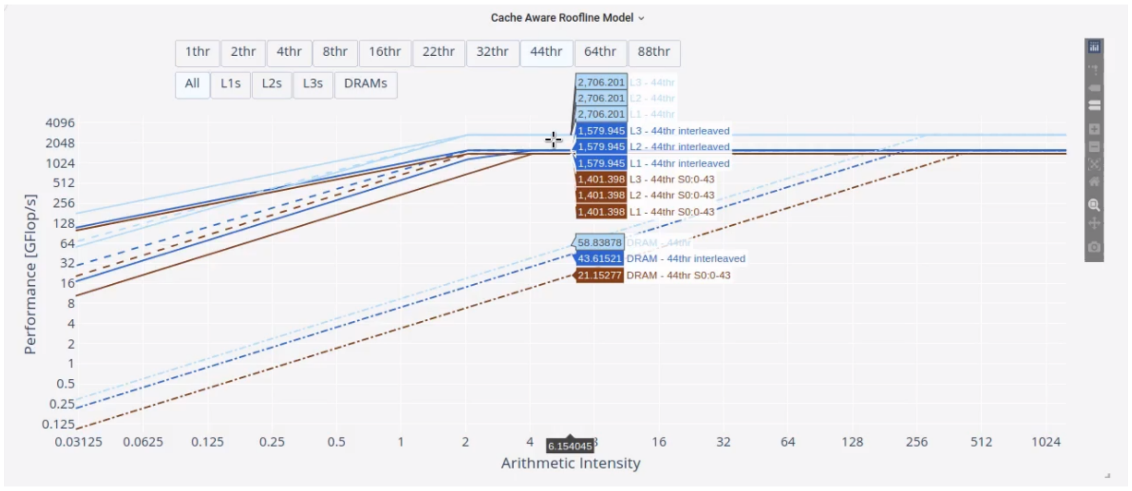


Figure 8 *When benchmark results are stored via BenchmarkInterface, environment modifiers or NUMA placement is saved alongside results. When these results are visualized, recorded environment settings are also presented.*

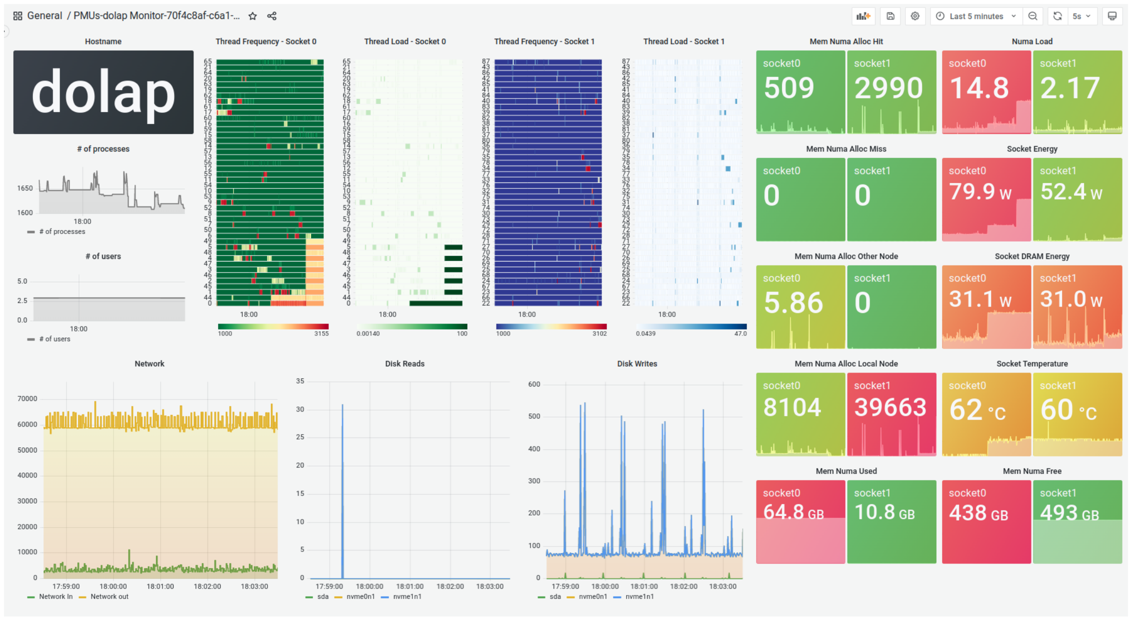


Figure 9 *A live monitoring dashboard that is tailored for machines that have multiple NUMA domains. As mentioned, metrics are paired with panels that match their value type and meaning best. Since topology information is also captured by STD, rows in thread load and thread frequency panels are sorted w.r.to shared cores automatically. This way, events that effect topologically close threads could be observed more clearly. For example, in this dashboard a load is placed in threads 0, 1, 2, 3, 4, 5, however frequency is also increased in threads that share core with this threads. Since the JSON object that translate into this dashboard is mutable, new metrics with desired panel type could be appended by user at any time.*

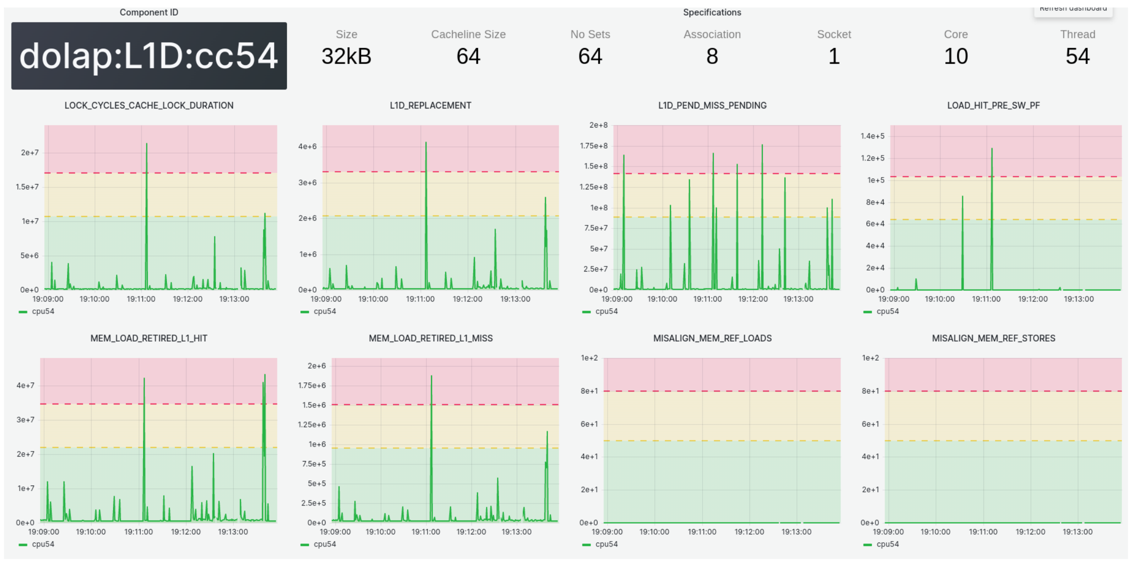


Figure 10 *A component dashboard that is generated for L1D cache on thread54. As mentioned, Interfaces that make STD could be isolated from each other and could be used to generate exclusive dashboards for the component. Dashboards for network, disks, sockets, cores, threads, L1, L2, L3 caches are generated automatically via filtering the metrics that the system report at the time. When the metrics collected from the system change, panels on these dashboards are also automatically changed. Specifications panel on this dashboard is generated with embedding Plotly indicators that shows static information that is encoded in STD. Also note that visualized metrics on this dashboard are PMU metrics that are normally collected when an execution takes place. However, they are set to be monitored constantly therefore provide a live view on the component.*

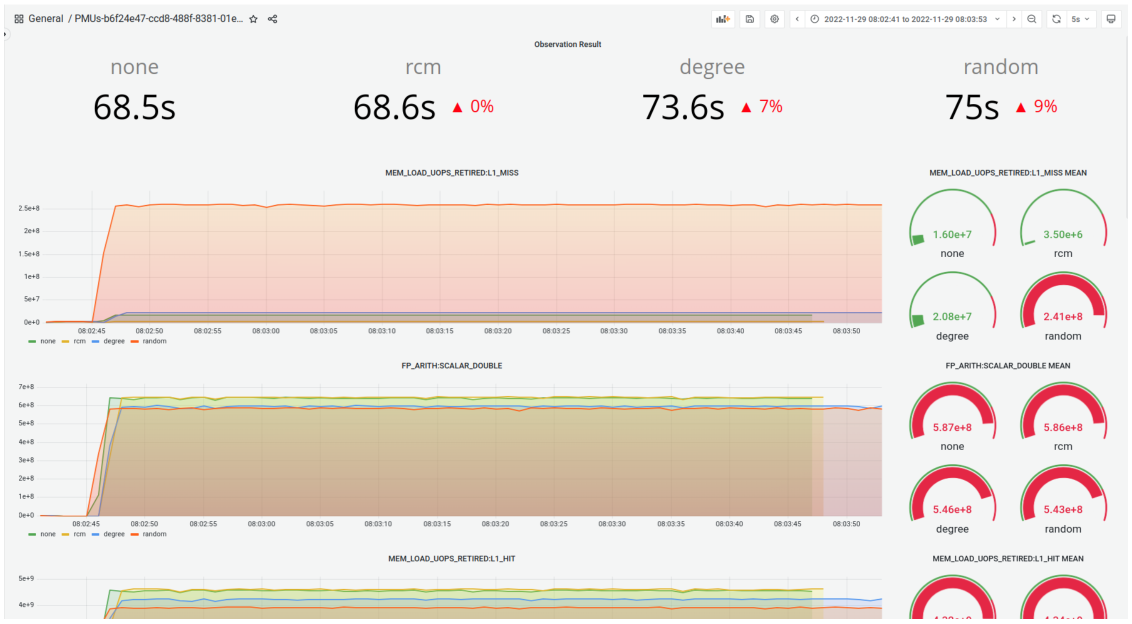

General / PMUs-b6f24e47-ccd8-488f-8381-01e...
2022-11-29 08:02:41 to 2022-11-29 08:03:53
Observation Result
none 68.5s
rcm 68.6s ▲ 0%
degree 73.6s ▲ 7%
random 75s ▲ 9%
MEM_LOAD_UOPS_RETIRED:L1_MISS
MEM_LOAD_UOPS_RETIRED:L1_MISS MEAN
1.60e+7 none
3.50e+6 rcm
2.08e+7 degree
2.41e+8 random
FP_ARITH:SCALAR_DOUBLE
FP_ARITH:SCALAR_DOUBLE MEAN
5.87e+8 none
5.86e+8 rcm
5.46e+8 degree
5.43e+8 random
MEM_LOAD_UOPS_RETIRED:L1_HIT
MEM_LOAD_UOPS_RETIRED:L1_HIT MEAN

Figure 11 *A comparison dashboard generated using four ObservationInterface that belong to the same system. In this figure, same SpMV kernel that use different orderings are compared. Collected metrics are originally from different times but their timestamps are synchronized in order to be able to observe program phases. This method will further developed with the addition of annotations that highlight program phases.*

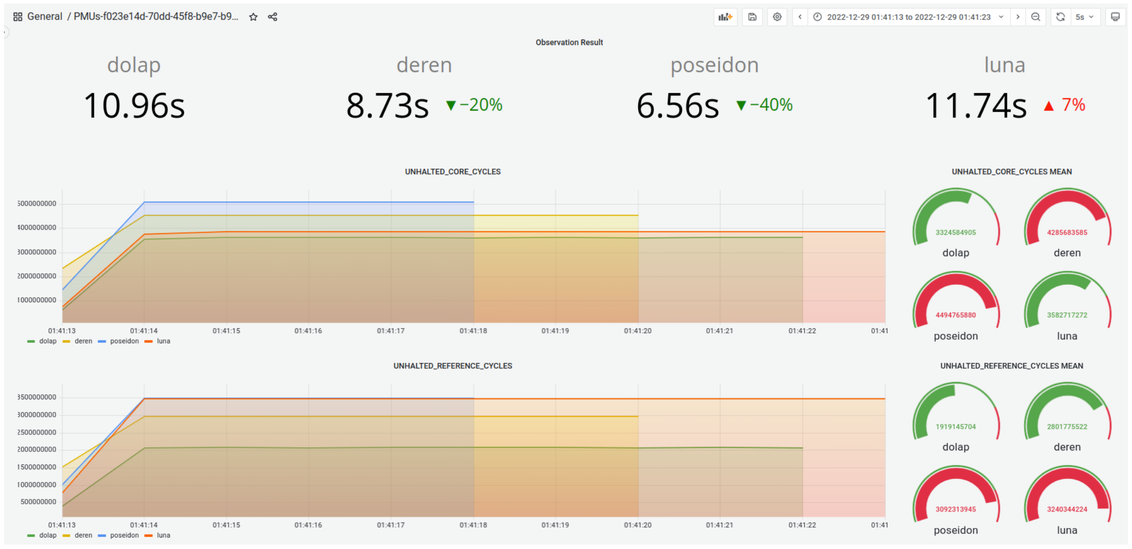
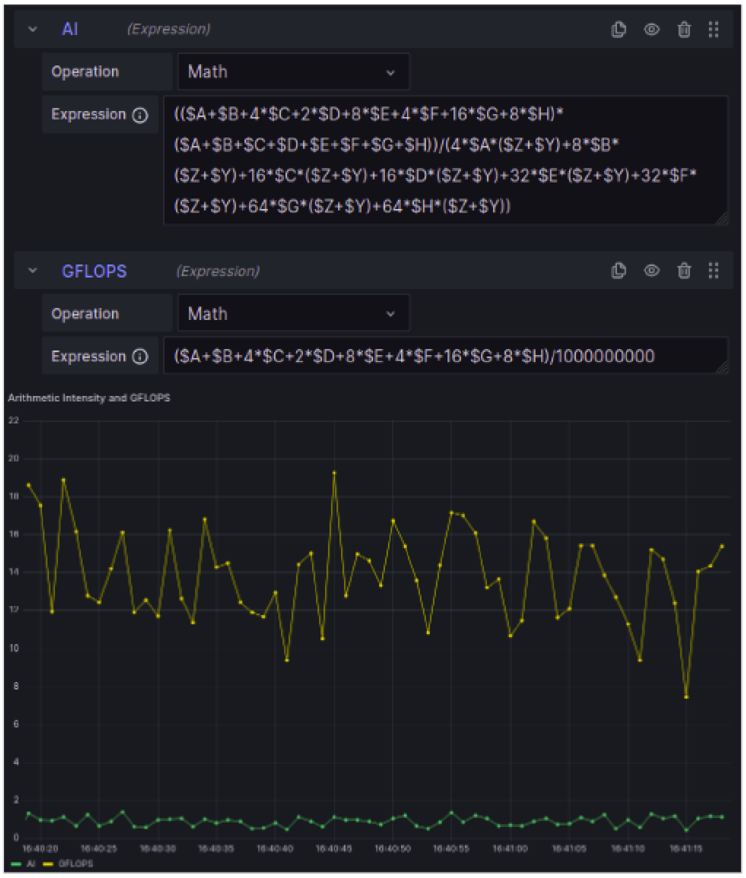


Figure 12 *A comparison dashboard generated using four ObservationInterface that belong to different systems. Since the ObservationInterface reported from different systems are homogeneous in syntax, they could compared instantly if they include same metrics. For this figure, all four systems are on the same network but the same comparison is possible for the systems that are in different networks even if their STDs are stored in different computers.*


AI (Expression)

Operation: Math

Expression:
```
(($A+$B+4*$C+2*$D+8*$E+4*$F+16*$G+8*$H)*
($A+$B+$C+$D+$E+$F+$G+$H))/(4*$A*($Z+$Y)+8*$B*
($Z+$Y)+16*$C*($Z+$Y)+16*$D*($Z+$Y)+32*$E*($Z+$Y)+32*$F*
($Z+$Y)+64*$G*($Z+$Y)+64*$H*($Z+$Y))
```

GFLOPS (Expression)

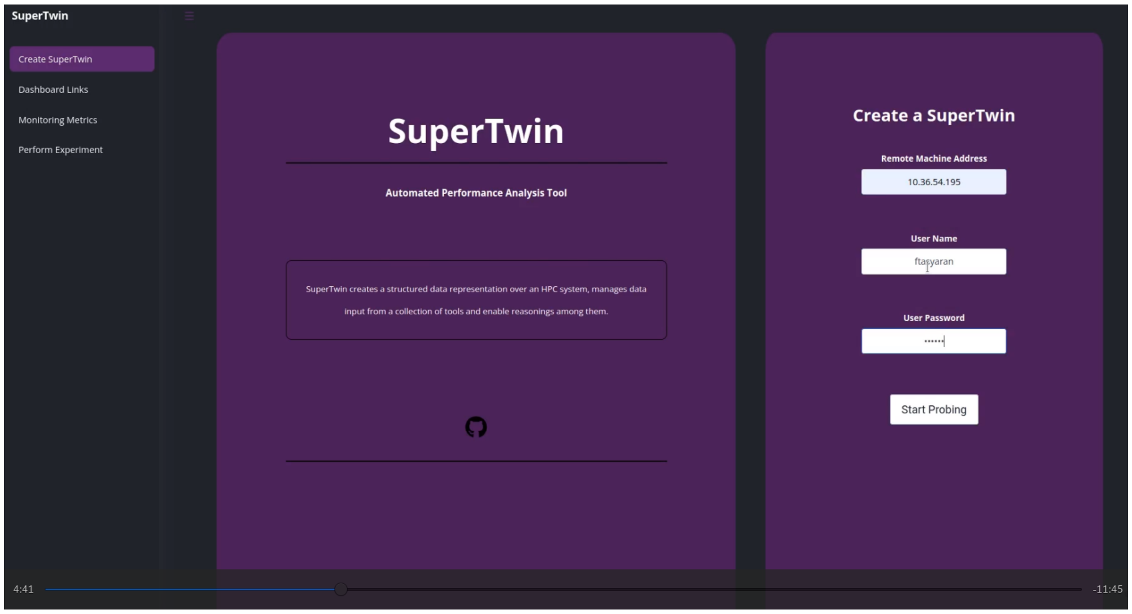
Operation: Math

Expression:
```
($A+$B+4*$C+2*$D+8*$E+4*$F+16*$G+8*$H)/1000000000
```

Figure 13 *Under development live Arithmetic Intensity dashboard. Since PCP can collect PMU metrics in real time, SuperTwin can visualize them to generate live dashboards and live performance models. In this figure, Arithmetic Intensity that is normalized w.r.to executed instructions vector size and memory throughput is presented. Equations to calculate arithmetic intensity and memory throughput are also automatically generated by SuperTwin and added to Grafana panels. This method will be further developed to provide real time performance models.*

2.5 SUPERTWIN WEB APPLICATION

Apart from Python API and command line interface, SuperTwin also implements a web application for a user-friendly usage. In the same manner as other SuperTwin functionalities, web application only requires STD of a target machine to operate. Web application provides configuration and launch functionalities for samplers and filtering for metrics to collect. This is useful since the number of metrics that could be collected from a target system is in the order of thousands, and encoded in STD which is hard to read for humans. Moreover, executions for a target system could be launched from web application with filtered and selected metrics with selected affinity. Web application also provides access to monitor and generated comparison dashboards, therefore effectively reduces the required number of applications/frameworks to use simultaneously to be able to use SuperTwin effectively to one.



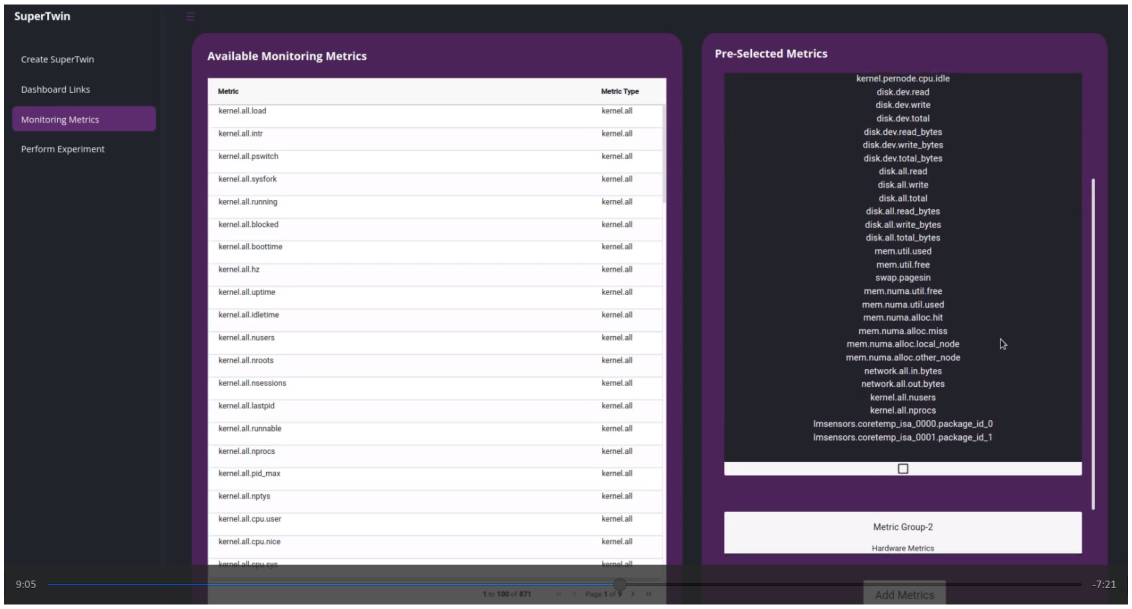
SuperTwin creates a structured data representation over an HPC system, manages data input from a collection of tools and enable reasonings among them.

Remote Machine Address: 10.36.54.195

User Name: ftasyaran

User Password: ••••••

Figure 14 *Login page of SuperTwin web application. When user log in with their account on the target system probing phase starts, STD and monitoring dashboard for target system is generated for the target system. Benchmarks are disabled by default due to their time requirements but could be set to launch automatically if user wants to generate roofline dashboards. After the first login, generated STD is retrieved for the target machine and SuperTwin functionalities are simply provided to user without any other operation.*



Available Monitoring Metrics

Metric	Metric Type
kernel.all.load	kernel.all
kernel.all.intr	kernel.all
kernel.all.pswitch	kernel.all
kernel.all.sysfork	kernel.all
kernel.all.running	kernel.all
kernel.all.blocked	kernel.all
kernel.all.boottime	kernel.all
kernel.all.hz	kernel.all
kernel.all.uptime	kernel.all
kernel.all.idletime	kernel.all
kernel.all.nusers	kernel.all
kernel.all.nroots	kernel.all
kernel.all.nsessions	kernel.all
kernel.all.lastpid	kernel.all
kernel.all.runnable	kernel.all
kernel.all.nprocs	kernel.all
kernel.all.pid_max	kernel.all
kernel.all.nptys	kernel.all
kernel.all.cpu.user	kernel.all
kernel.all.cpu.nice	kernel.all
kernel.all.cpu.sys	kernel.all

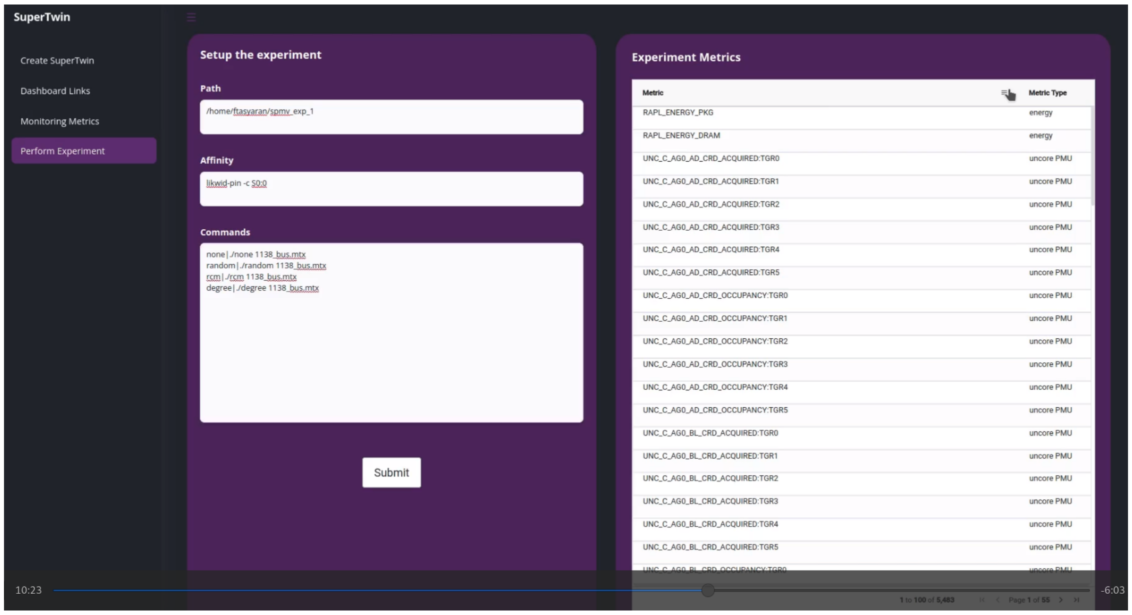
1 to 100 of 871 Page 1 of 9

Pre-Selected Metrics

kernel.pernode.cpu.idle
disk.dev.read
disk.dev.write
disk.dev.total
disk.dev.read_bytes
disk.dev.write_bytes
disk.dev.total_bytes
disk.all.read
disk.all.write
disk.all.total
disk.all.read_bytes
disk.all.write_bytes
disk.all.total_bytes
mem.util.used
mem.util.free
swap.pagesin
mem.numa.util.free
mem.numa.util.used
mem.numa.alloc.hit
mem.numa.alloc.miss
mem.numa.alloc.local_node
mem.numa.alloc.other_node
network.all.in.bytes
network.all.out.bytes
kernel.all.nusers
kernel.all.nprocs
lmsensors.coretemp_isa_0000.package_id_0
lmsensors.coretemp_isa_0001.package_id_1

Metric Group-2
Hardware Metrics

Figure 15 *Metric selection page of web application. In this page, suggested metrics could be chosen as a chunk, or user can view and add any of the metrics that the target system can report. When a metric is added from this page, it's automatically appended to the target system's monitoring dashboard.*



Setup the experiment

Path: /home/ftasyaran/spmv_exp_1

Affinity: likwid-pin -c S0:0

Commands:

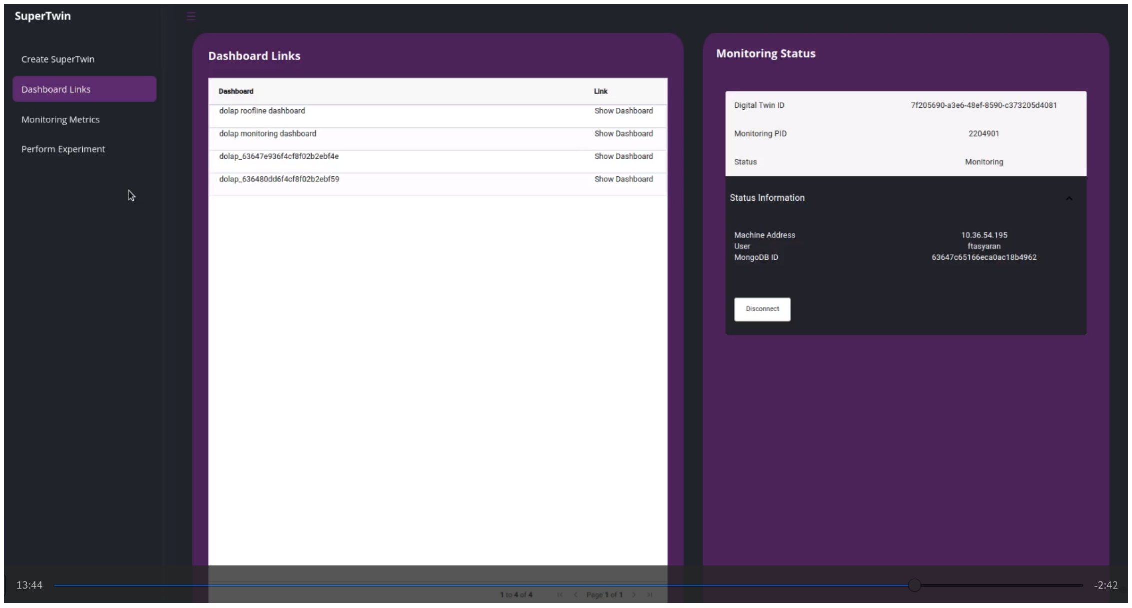
```
none|./none 1138_bus.mtx
random|./random 1138_bus.mtx
rcm|./rcm 1138_bus.mtx
degree|./degree 1138_bus.mtx
```

Experiment Metrics

Metric	Metric Type
RAPL_ENERGY_PKG	energy
RAPL_ENERGY_DRAM	energy
UNC_C_AG0_AD_CRD_ACQUIRED:TGR0	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR1	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR2	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR3	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR4	uncore PMU
UNC_C_AG0_AD_CRD_ACQUIRED:TGR5	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR0	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR1	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR2	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR3	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR4	uncore PMU
UNC_C_AG0_AD_CRD_OCCUPANCY:TGR5	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR0	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR1	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR2	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR3	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR4	uncore PMU
UNC_C_AG0_BL_CRD_ACQUIRED:TGR5	uncore PMU
UNC_C_AG0_BL_CRD_OCCUPANCY:TGR0	uncore PMU

1 to 100 of 5,483 Page 1 of 55

Figure 16 *Perform experiment page of the web interface. From this page, executions on target system could be launched with selected path, application and affinity. PMU metrics to be collected from the target system during the execution could also be filtered and selected from this page. A script is then generated and executed automatically for the each line written to commands box. A dashboard that includes comparison of the given commands, similar to Figure 11 is then generated.*



Dashboard Links

Dashboard	Link
dolap roofline dashboard	Show Dashboard
dolap monitoring dashboard	Show Dashboard
dolap_63647e936f4cf8f02b2ebf4e	Show Dashboard
dolap_636480dd6f4cf8f02b2ebf59	Show Dashboard

Monitoring Status

Digital Twin ID	7f205690-a3e6-48ef-8590-c373205d4081
Monitoring PID	2204901
Status	Monitoring

Status Information

Machine Address	10.36.54.195
User	ftasyaran
MongoDB ID	63647c65166eca0ac18b4962

Figure 17 *Monitoring, performance model, observation and other generated dashboards for the target system could be accessed at any time from dashboards page. Status of metric collection could also be viewed and dead samplers (if any) could be recovered from this page.*

2.6 CONTRIBUTIONS BY EACH PARTNER

INESC-ID provided the cache-aware roofline model to blend into SuperTwin and worked with SU on the implementation of benchmarks within SuperTwin. SU and KU have worked in SparseBase together. KU also provided ReuseTracker and ComDetective.

2.7 DEVIATIONS (IF ANY)

Except for Task 4.6, we are progressing according to the plan. We have already started Task 4.6, designed a performance database, but paused its implementation since the structure heavily depends on the outputs of the other tasks. We will focus on the implementation part in the remaining project period.

REFERENCES

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.

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

Janowicz. "SOSA: A lightweight ontology for sensors, observations, samples, and actuators". *Journal of Web Semantics* 56 (2019), pp. 1–10. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2018.06.003. URL: https://www.sciencedirect.com/science/article/pii/S1570826818300295.

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

3 HISTORY OF CHANGES

Version	Author(s)	Date	Comment
0.1	Fatih Taşyaran	25.06.2023	First draft
0.2	Kamer Kaya	30.06.2023	Modifications
0.2.1	Didem Unat	30.06.2023	Final version

Table 2 *Document History of Changes*