Technische Universität Dresden

Maximilian Knespel[1], Bert Wesarg[1+2], Mikhail Zarubin[1]
[1]TU Dresden / CIDS / ZIH, [2]GWT-TUD GmbH
16th International Parallel Tools Workshop 2025, Stuttgart

Scalable Metric Calculations Across Hardware Levels in Vampir

Outline

- Motivation
- Hardware Hierarchy in Score-P
- Extended Metric Plugin API for Score-P/lo2s
- Scalable Metric Calculations in Vampir

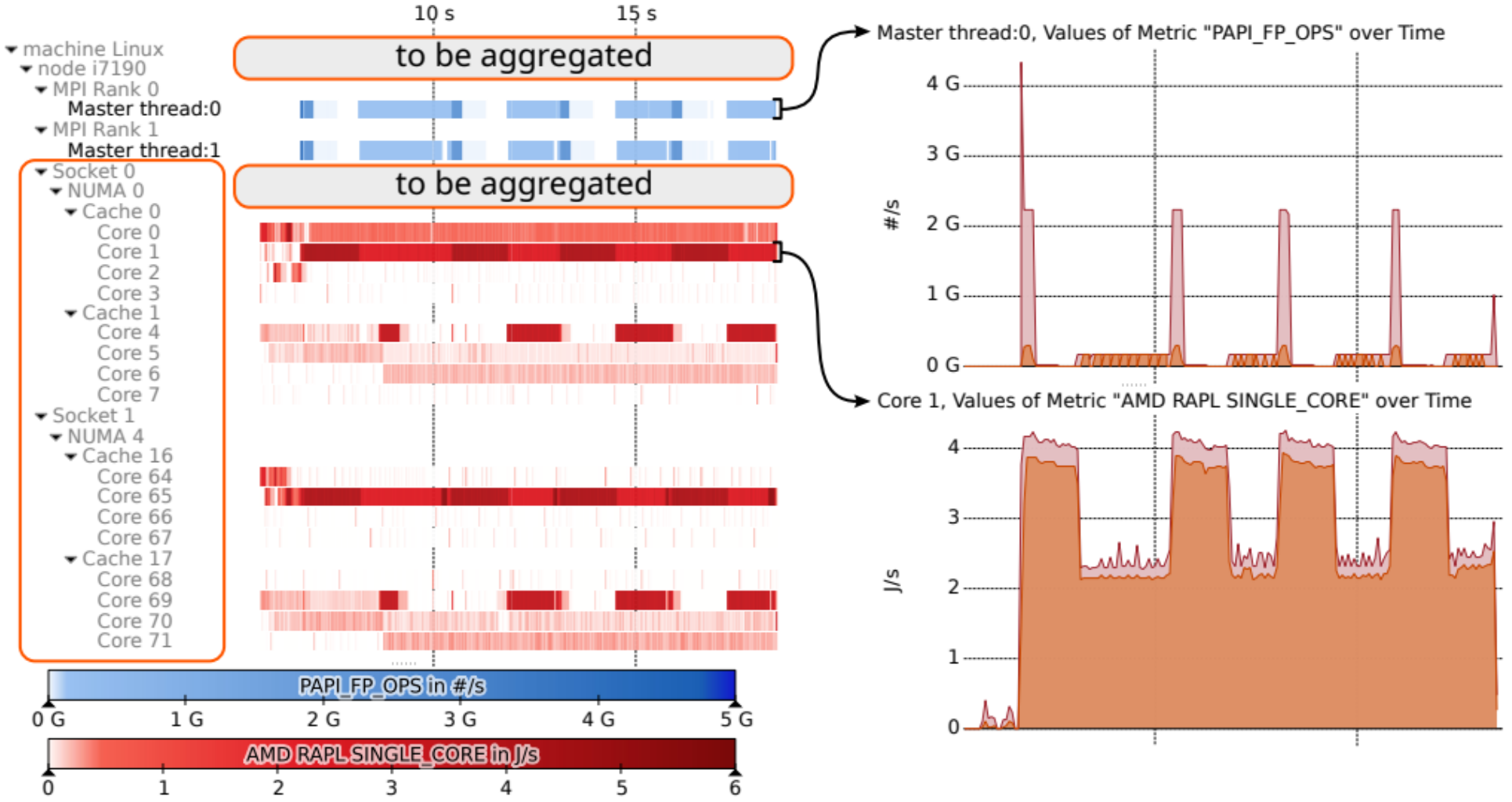
Motivation

Aggregate energy usage (in kWh or J), or the total FLOPs over all traced locations and compute that energy/FLOP for all of the used cluster

- Program Execution
- Application Tracing: Score-P, lo2s, …
- **Tracing Plugin: Score-P power and energy event plugin counters, ...**
- Trace File Format: OTF2 (no changes necessary)
- **Scalable Trace Analysis: Vampir**

Goal: Extend the scalable analysis that can be applied after trace recording

Metric Visualization in Vampir



Goal

Given:

- Time-value pairs for power for each core and FLOPS for each thread
- A target location: node i7190 (or machine)

Compute the energy per FLOP by:

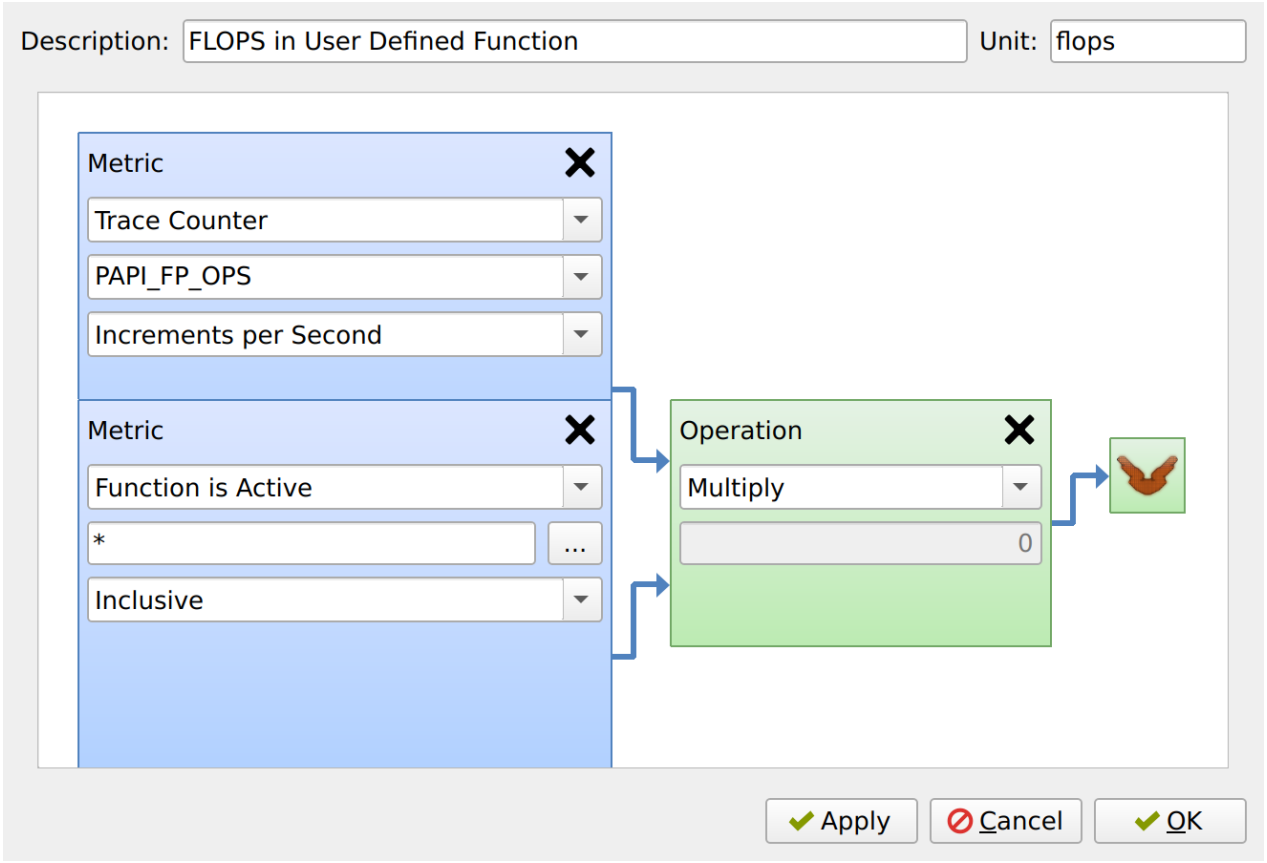
1) For all descendant locations of the target location, sum up the monotonically increasing FLOPs and energy counters
2) Differentiate the sum-aggregated timelines to get Watt per node (or machine) and FLOPS (FLOPs per second)
3) Divide the timelines to get performance (FLOPS) per Watt

Metric Expressions in Vampir

- A generic metric expression language to combine various events already exists
- This expression tree is only applied location-wise because of the distributed nature of Vampir's architecture

→A new cross-location operation is needed.

→ Specifying all input locations is cumbersome and might lead to complex communication patterns



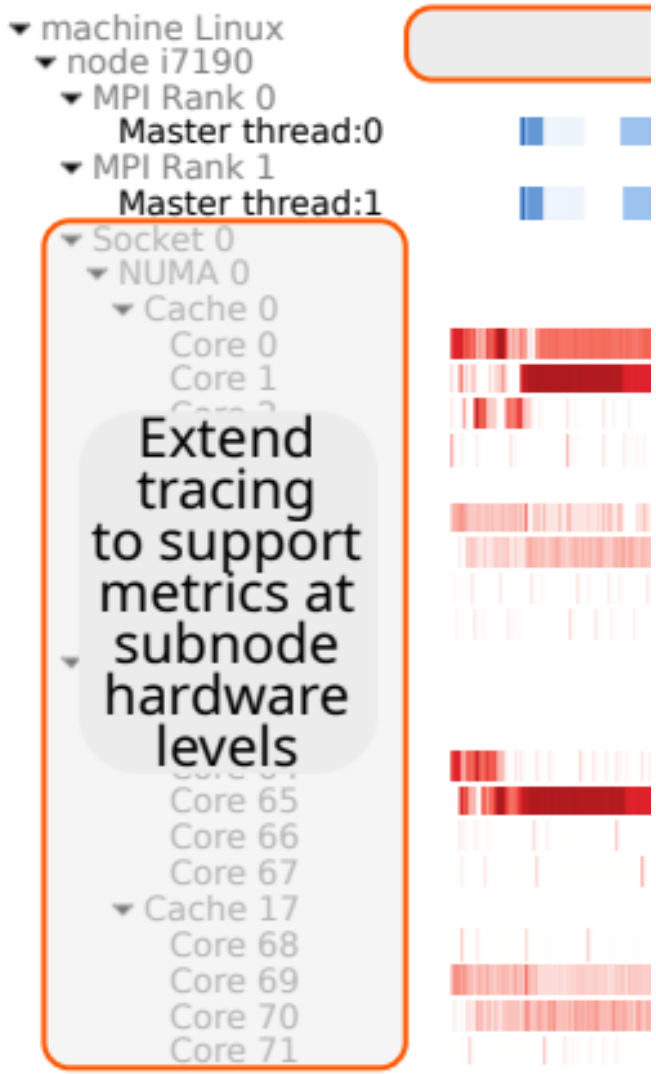
→ Introduce an "Aggregate" operation leveraging the hardware hierarchy ⇒ Score-P

Metric Plugin API

- lo2s and Score-P share a common metric plugin API, which serves as the basis for a broad range of system-specific plugins
- Metric recording is limited to:
  - Machine level
  - Node level
  - Process level
  - Thread level

    ⇒ All metrics are attributed to this level

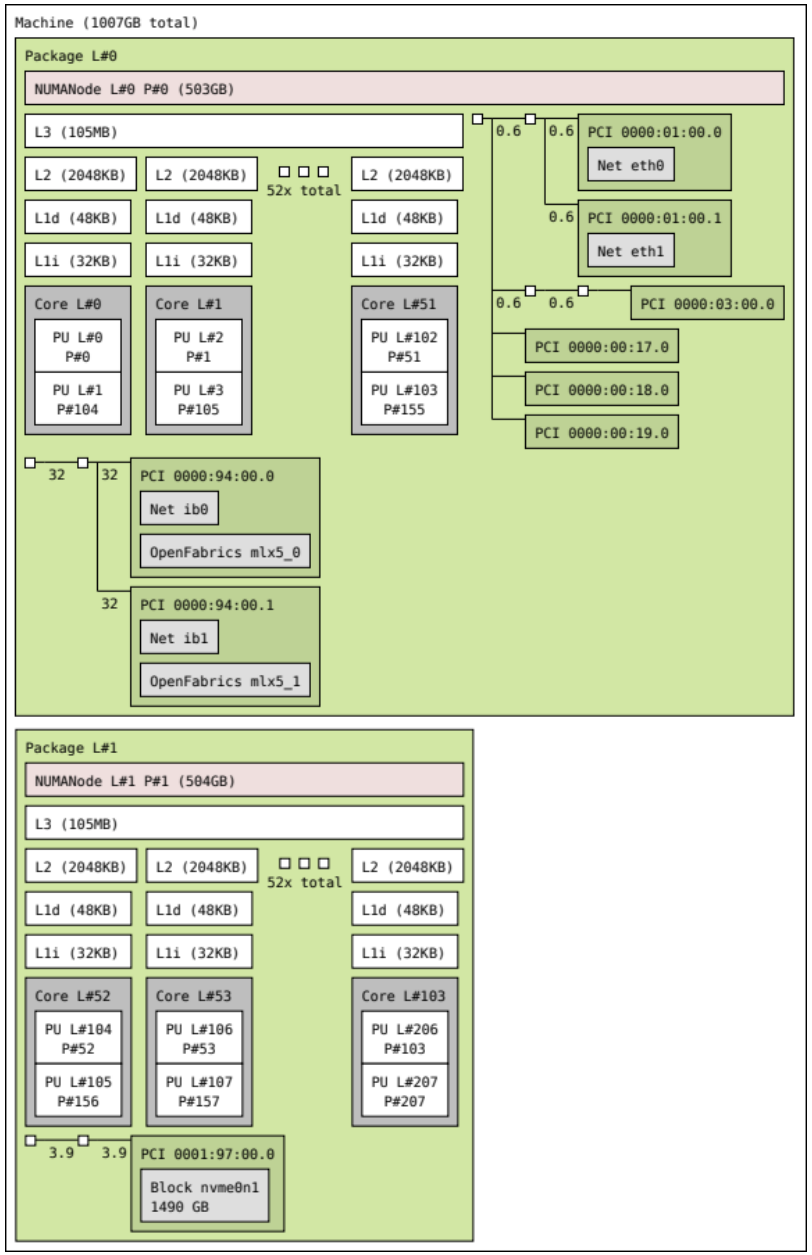
⇒ System tree and plugin API needs to be extended to allow proper attribution of metric measurement points



Portable Hardware Locality: hwloc

- Goal: simplify the process of discovering hardware resources in parallel architectures
- Software provides portable command-line tools and a C API for consulting these resources, their locality, attributes, and interconnections
- Obtains the hierarchical map of processing elements within a compute node
- Involved elements: NUMA memory nodes, shared caches, processor packages, cores, processing units (logical processors or "threads"), memories, PCI devices, GPUs, …

https://www.open-mpi.org/projects/hwloc/



Integrating hwloc into Score-P

- Since hwloc primarily aims at helping high-performance computing (HPC) applications and is part of Open MPI, it is a natural choice for Score-P integration.
  - Use *embedded* building mode to prefix all `hwloc_` functions with `SCOREP_Libhwloc_` instead, so that it will not clash with the hwloc version used by the traced user code or other runtimes/libraries
- Runs of single-child chains are merged into one node to simplify the hierarchy, e.g., cores, L1i, L2d, and L2 caches
- PCI bridges are skipped for networking and accelerator devices
- Because Score-P is a multi-process measurement system, a *conflict resolution* arbitration process is implemented using an all-to-all communication of all collected system tree nodes across all participating processes
  - The first in the sorted set of processes owning the node is responsible for any metrics ⇒ each node in all trees is marked *is-responsible* only once

Metric Plugin API Primer

Different recording modes available:

- (Strict) synchronous → Reads metrics at every/most Enter/Leave event
- Asynchronous → Periodic sampling of metrics

Broken down plugin protocol:

1. Operator (lo2s or Score-P) loads and inits plugin
2. **`get_event_info(<token>)`** Called once for each user provided *<token>* to get list of possible metrics to record. Includes name and metadata such as value type, unit, …
3. **`add_counter(<metric-name>)`** Called once for each metric if this process is responsible for the metric (*machine*, *host*), returns `<metric-id>`
4. **`synchronize(<point>)`**: Called at the *start* and *end* of the measurement
5. **`get_all_values(<metric-id>)`** Called for each metric once, returns a *(timestamp, value)* list

Extended Metric Plugin API

Problems:

- Metrics with multiple measurement points need a unique name , so that **add_counter(<metric-name>)** works as expected ("**[<index>]"** suffix)
- All measurement points are on the host in the system tree

Changed plugin protocol:

1. Added new *topology* level
2. **add_counter(<metric-name>)** unchanged, but no need for made up unique metric names
3. Instead of **add_counter(<metric-name>)** a new **add_topology_counters(<topology>, <metric-name>)** is called
   - *Topology* is a simple first-child/next-sibling linked tree structure with a type, a polymorphic ID, and the *is-responsible* field
   - Returns a *(<metric-id>, <node>)* list for all matched and *is-responsible* nodes

Extended Metric Plugin API: Adaptations

Converted two metric plugins:

1. https://github.com/score-p/scorep_plugin_x86_energy
2. https://github.com/score-p/scorep_plugin_rocm_smi

`x86_energy`

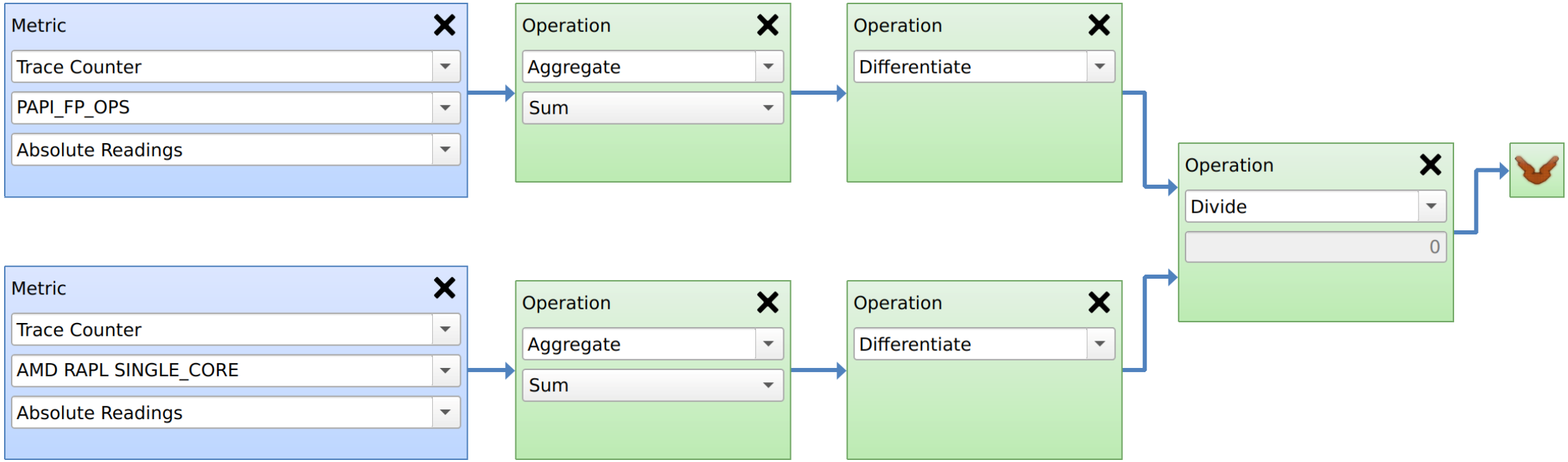
- Measures RAPL metrics on packages, cores, …
- Polymorphic node ID for such types contains OS index of entity

`rocm_smi`

- Measures GPU metrics via ROCm SMI API
- Polymorphic node ID for PCI types contains the PCI BDF of the device

→ Score-P provides sub-node hardware hierarchy with proper metric attribution ⇒ Vampir

Vampir's Aggregate Operation



The operation tree is sent as a reverse polish notation expression to the server:

```
"2" "stairstep" source-counter "sum" aggregate "0" diff-quot
"4" "stairstep" source-counter "sum" aggregate "0" diff-quot /
```

Requirements

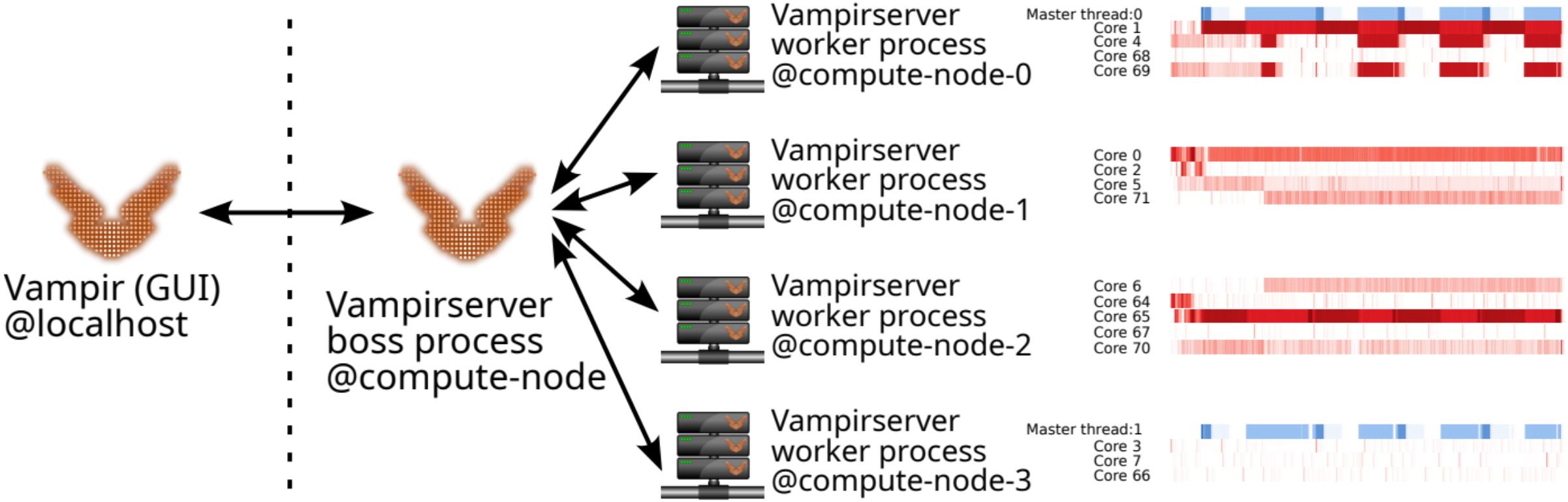
Scalability:

- Use Vampir architecture to process locations in parallel as best as possible
- Necessary communication should avoid sending all trace events through the network for each query

Simplifications:

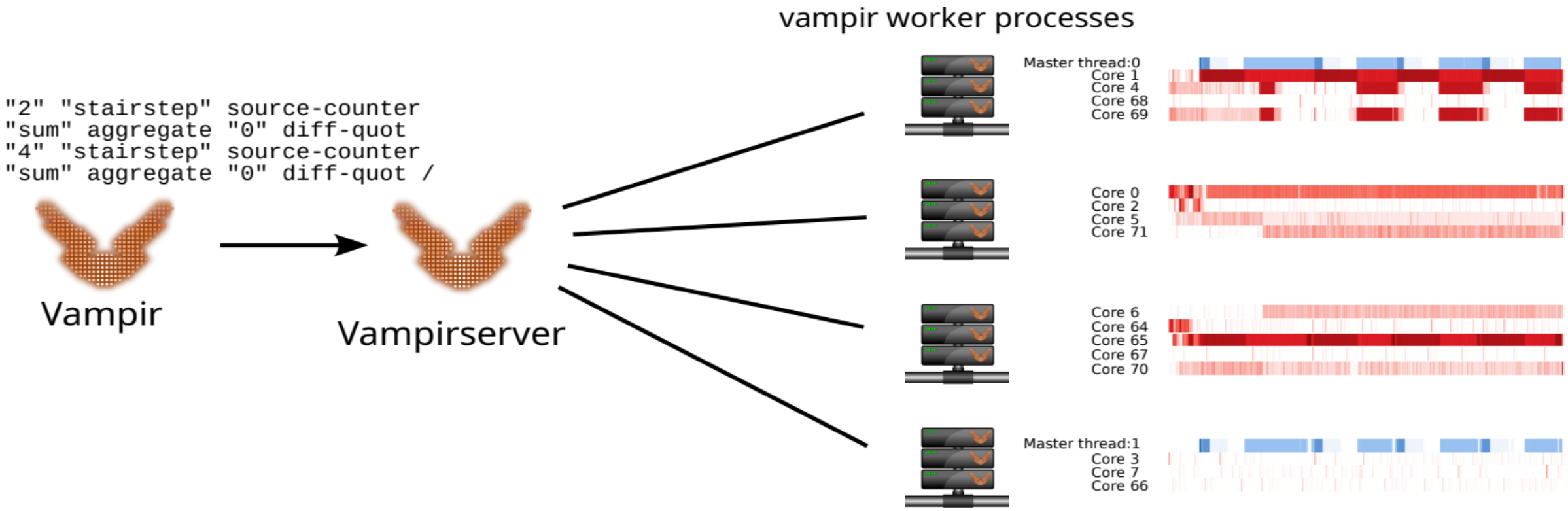
- Only aggregate monotonically increasing timelines because these suffer less from aliasing / sampling errors

Vampir Architecture



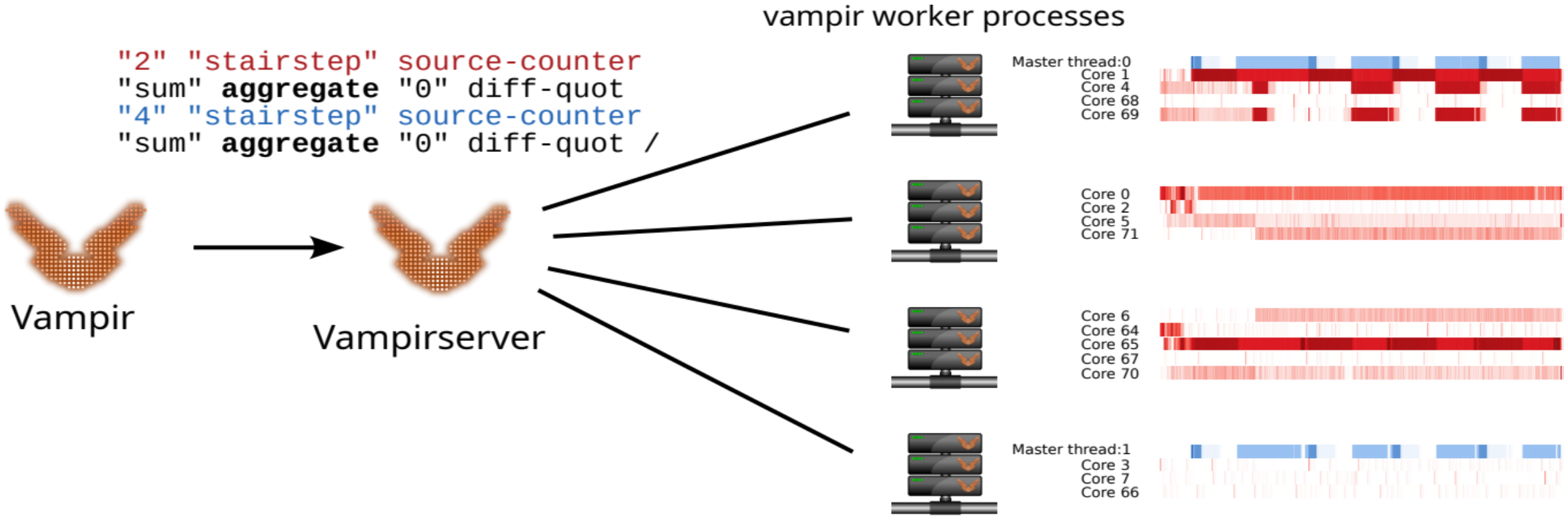
Query Computation

1. Send metric expression and requested target location to Vampirserver



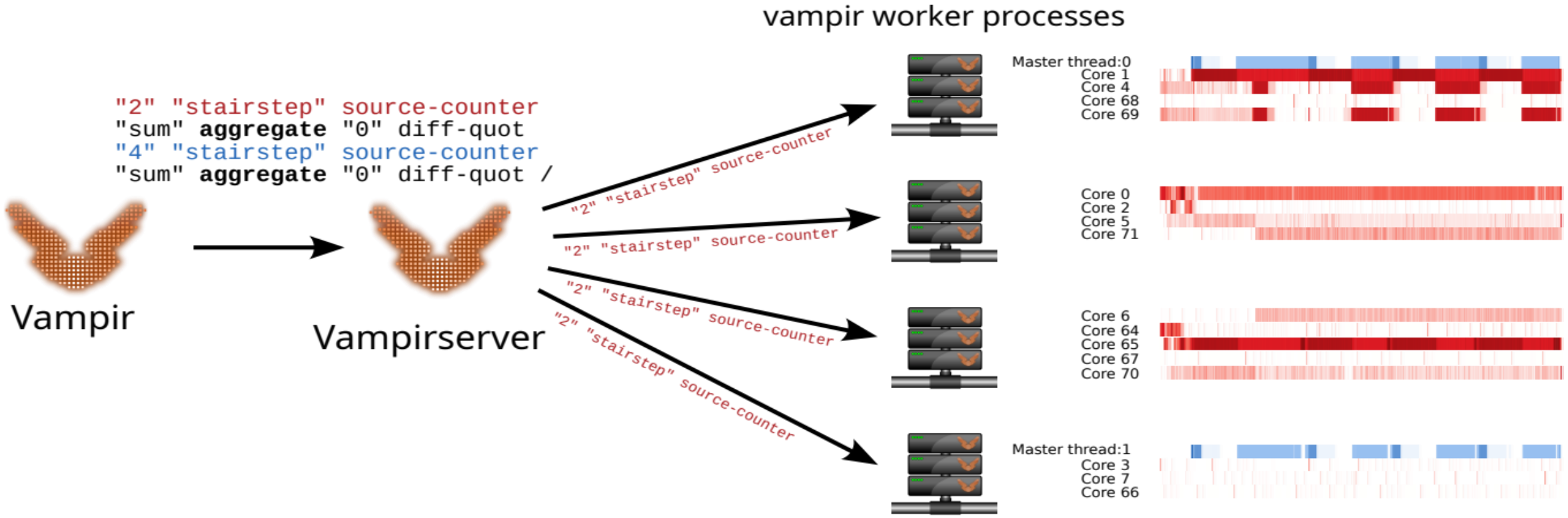
Query Computation

2. Look for aggregate operations in metric expression and extract the operation subtrees / subexpressions



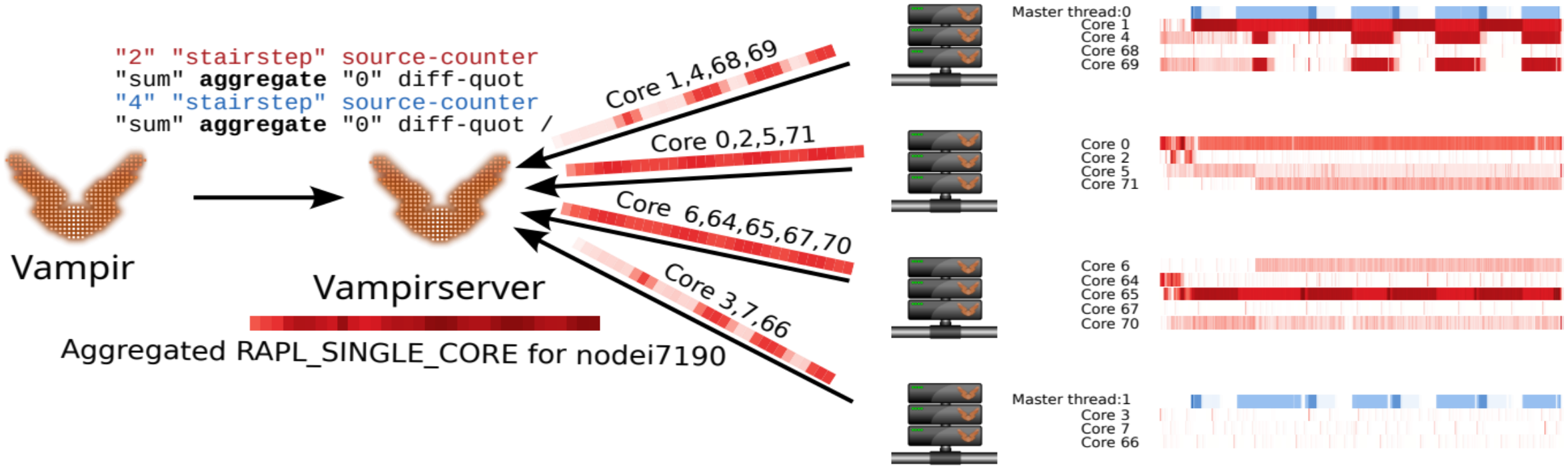
Query Computation

3. Aggregate the timeline for all descendants of the target location by requesting timelines from the workers and merging them



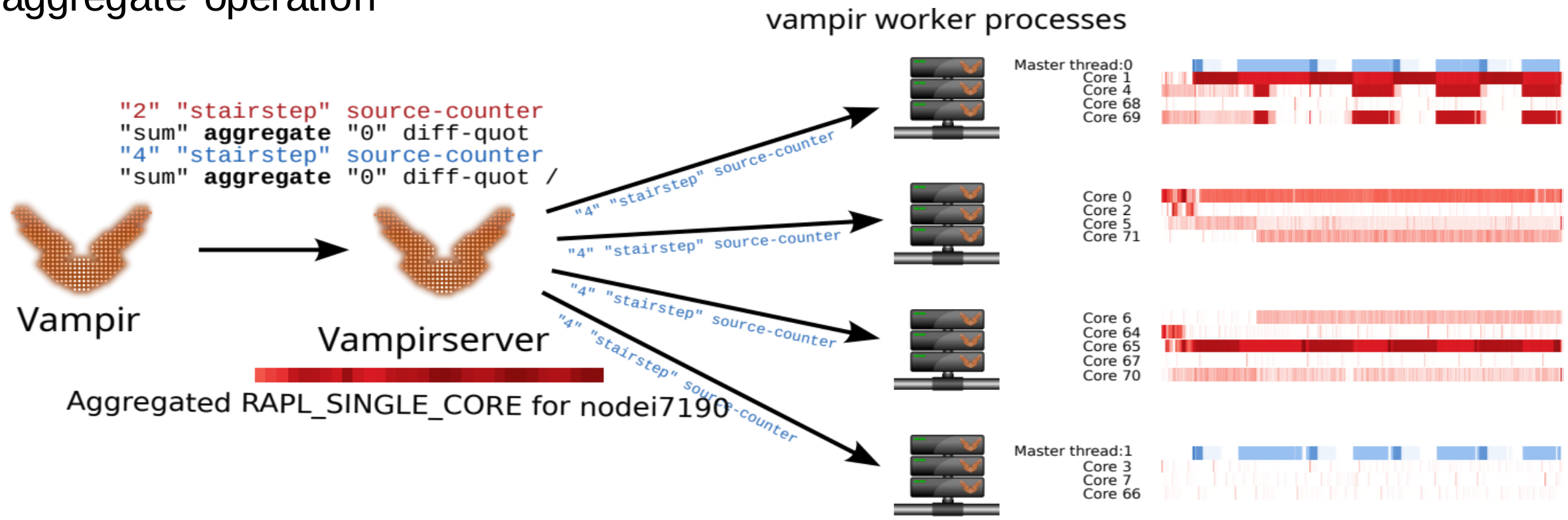
Query Computation

4. Merge the merged timelines into one timeline for the target location



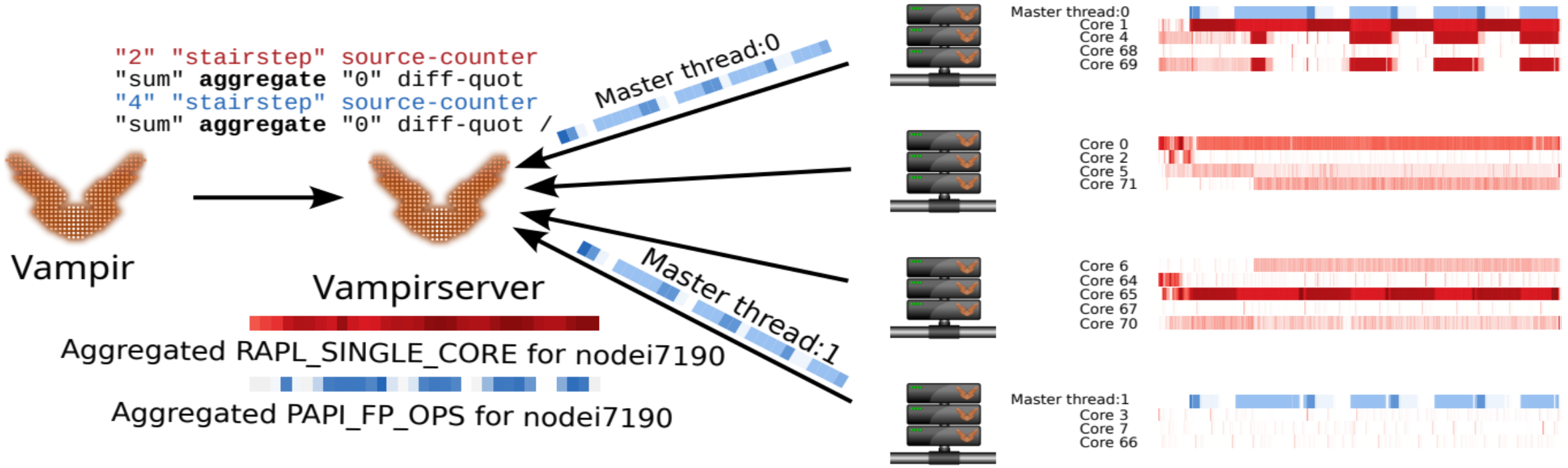
Query Computation

5. Repeat for the second subexpression (PAPI_FP_OPS), the input to the second aggregate operation



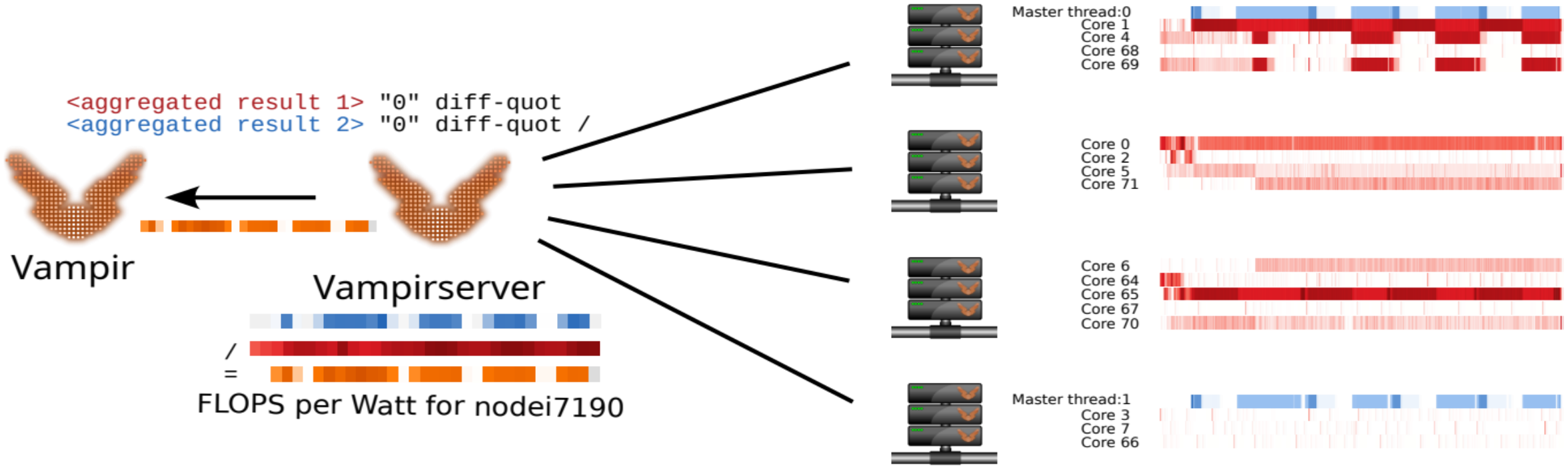
Query Computation

5. Repeat for the second subexpression (PAPI_FP_OPS), the input to the second aggregate operation

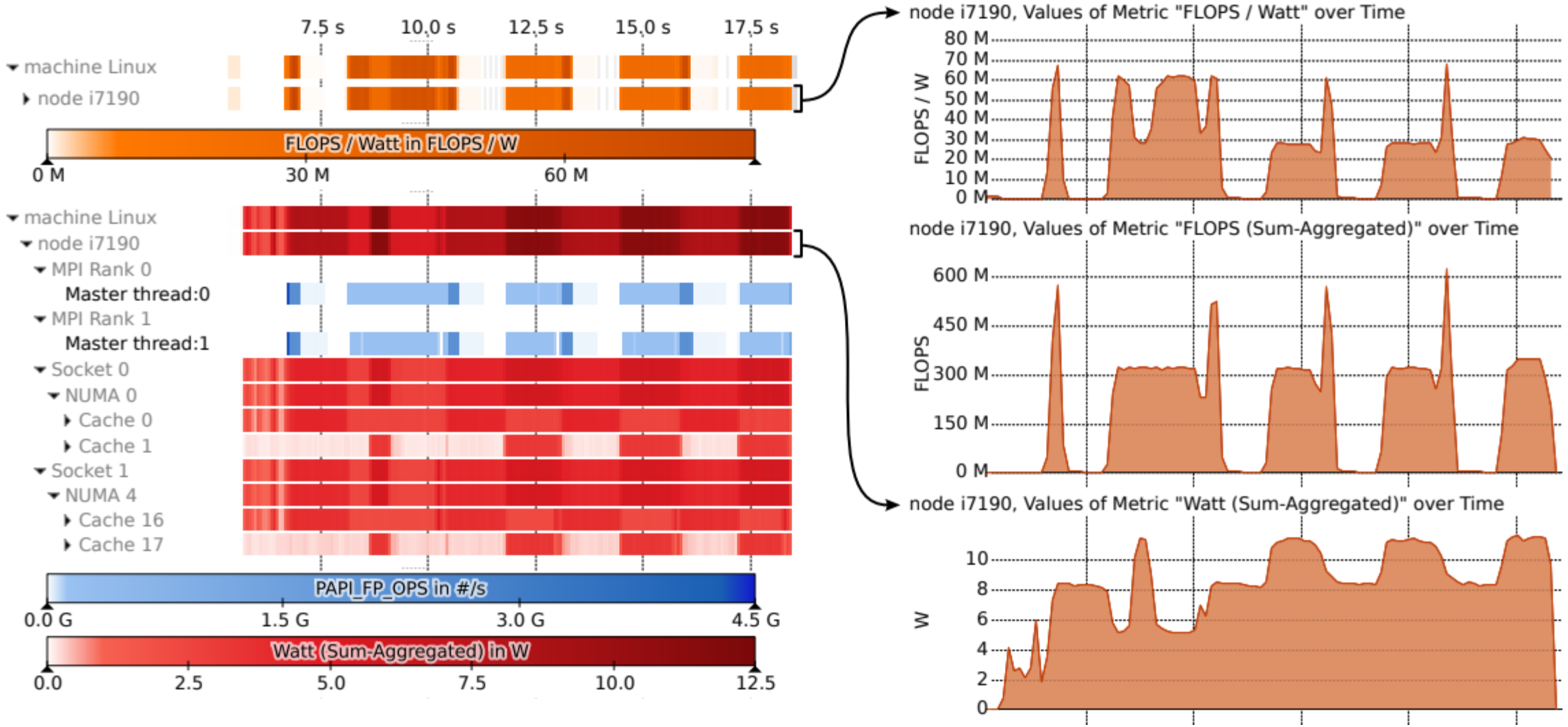


Query Computation

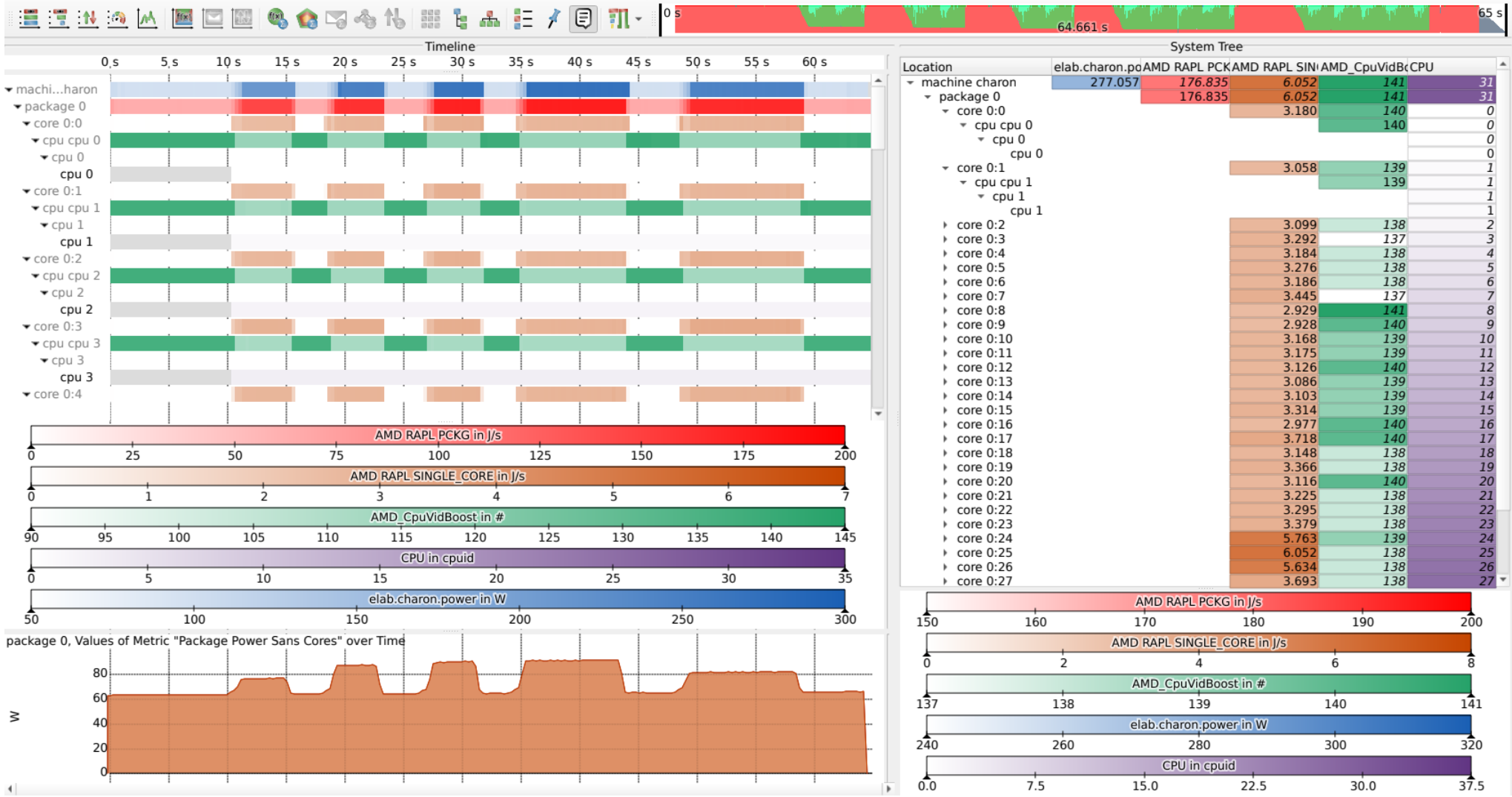
6. Compute outer expression (derive by time and divide it) using the intermediary results and return the overall result



Results

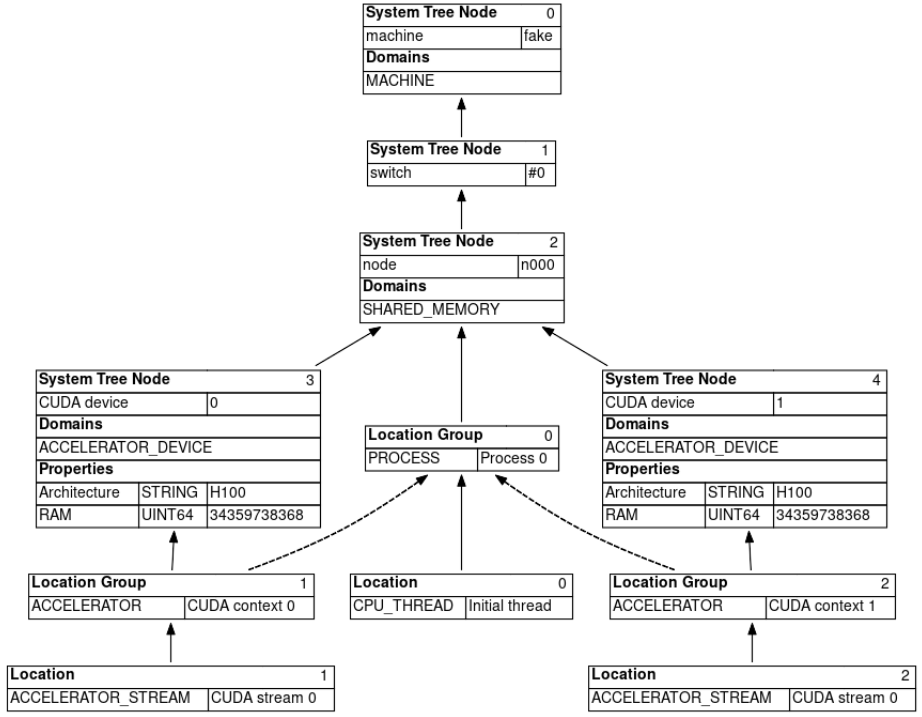

7.5 s
10.0 s
12.5 s
15.0 s
17.5 s
machine Linux
node i7190
FLOPS / Watt in FLOPS / W
0 M
30 M
60 M
machine Linux
node i7190
MPI Rank 0
Master thread:0
MPI Rank 1
Master thread:1
Socket 0
NUMA 0
Cache 0
Cache 1
Socket 1
NUMA 4
Cache 16
Cache 17
PAPI_FP_OPS in #/s
0.0 G
1.5 G
3.0 G
4.5 G
Watt (Sum-Aggregated) in W
0.0
2.5
5.0
7.5
10.0
12.5
node i7190, Values of Metric "FLOPS / Watt" over Time
FLOPS / W
80 M
70 M
60 M
50 M
40 M
30 M
20 M
10 M
0 M
node i7190, Values of Metric "FLOPS (Sum-Aggregated)" over Time
FLOPS
600 M
450 M
300 M
150 M
0 M
node i7190, Values of Metric "Watt (Sum-Aggregated)" over Time
W
10
8
6
4
2
0

Resulting lo2s Trace



Other Improvements

- The Performance Radar now can be hierarchically grouped just like the Master Timeline
- The Performance Radar, System Tree, and Master Timeline can now show multiple metrics
  - The timelines only allow multiple metrics as long as their measuring points do not overlap.
  - Improve Graphviz DOT file output of `otf2-print --system-tree` and add the list of recorded metrics per location



Summary

- Added support for subnode system tree levels and hwloc to Score-P metric plugin interface
- Extended energy plugin to attribute counters on the correct system tree node
- Added new aggregate operations to the Vampir metric expression language
- Use the system tree to implement those aggregate operations
- Implement a multi-staged algorithm in Vampirserver with a focus on upper bounds for the amount of data to be communicated between each stage

Outlook

- Look into smarter trade-offs between communication message size and aliasing errors
- Look into alternative bounded message size schemes, e.g., Fourier coefficients, established compression algorithm, ...
- Generalize the concept to nested aggregate operations aggregating over user-defined system tree levels
- Automatic smart aggregation (collapse node ⇒ aggregation of all children)
- Extend plugin API improvements to supra-node hierarchy, to e.g., record energy metrics for a rack
- Extend system tree structure with cross edges between nodes, allows to record metrics for the links, e.g., XGMI/NVLink throughput between GPUs, NUMA, IB, …