sys-sage: A Fresh View on Dynamic Topologies & Attributes of HPC Systems

Stepan Vanecek
stepan.vanecek@tum.de
Technical University of Munich, Garching, Germany
Garching bei München, Germany

Martin Schulz
schulzm@in.tum.de
Technical University of Munich, Garching, Germany
Garching bei München, Germany

ABSTRACT

HPC systems are getting ever more powerful, but this comes at the price of increasing system complexity. In order to use HPC systems efficiently, one has to be aware of their architectural details, in particular details of their hardware topology, which is increasingly affected by dynamic runtime settings.

sys-sage is a novel approach providing an infrastructure for storage, correlation, and provision of HW-related system information. It uses information from various well-known sources as well as use-case-specific solutions, and correlates the particular pieces together to provide a full view of a system. The novelty of our approach lies in the ability to capture dynamic environments as well as systems' complexities, and in enabling greater flexibility in its usage.

sys-sage is publicly available, and can be used by many applications. It integrates widely used approaches, such as hwloc or dynamic counter information, and offers user-integration of all other user-specific data sources.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; • **Computing methodologies** → *Modeling and simulation*; *Parallel computing methodologies.*

KEYWORDS

HPC System Topology, Hardware Architecture, Heterogeneous Computing, Performance Optimizations.

ACM Reference Format:
Stepan Vanecek and Martin Schulz. 2018. sys-sage: A Fresh View on Dynamic Topologies & Attributes of HPC Systems . In *Proceedings of SC Research Poster (SC '23).* ACM, New York, NY, USA, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 MOTIVATION

Modern High-Performance Computing (HPC) architectures have evolved into clusters of complex and heterogeneous multi-core processing units. This design shift has led to increased architectural complexity of both chip and node designs. The traditional static and strictly hierarchical representation of such systems (as presented, e.g., by hwloc [1]) does not provide sufficient information to fully understand such systems anymore.

Due to this trend, being able to fully utilize the systems and having an understanding of systems' behavior has become a challenge. Having a deep understanding of the architecture and properties of the system, including its dynamic abilities as well as data transfer capabilities, is very important for efficient parallel application design, performance tuning for a given system, performance management, resource sharing, as well as scheduling and data allocation decisions, to only name a few areas. As a consequence, there is a need for a solution combining the strengths of the simple-to-interpret hwloc data view with support of non-hierarchical and dynamic data. This approach has to go beyond the static hwloc view and extend it with a more dynamic view representing the system's data flow capabilities as well as its dynamic configuration abilities.

2 TARGET USAGE

A wide range of applications and use-cases need some kind of information regarding a system's topology, configuration, or capabilities. Current solutions only provide a specific subset of the required information, especially considering the challenges posed by modern systems. We aim to provide the necessary infrastructure for supporting any use-case dealing with HW-related data.

The use-cases we address with our library span across many different disciplines and areas of HPC. To list a few examples, we consider tasks regarding scheduling nodes based on their mutual connectivity (bandwidth/latency), scheduling threads based on proximity (to each other, to a special memory, or to a GPU), or scheduling on power-efficiency vs. performance cores. We also address the needs for resource-sharing tasks, i.e., how to split a component (a node or a single CPU/GPU) and decide which resources to offer to which application (e.g., allocate an application on a socket close to a particular GPU). Additionally, we also target heterogeneous and deep memory systems: in order to decide on which memory type to allocate data, knowledge of bandwidth/latency, size, or configuration for each option is required. Further, we need to consider power-management use-cases that make decisions based on the cost (in terms of power) of data transfers between particular components. Finally, we also support advanced performance modeling tools that simulate how a system would behave if it had different characteristics (more/fewer cores, higher/lower bandwidth, etc.).

Different applications have widely different requirements regarding what information such a library should provide, i.e., what system-relevant information they require and what is redundant. We design a modular approach that decouples the core tasks to gain flexibility for such a broad coverage. For this, we split the workflow into three stages:

SC '23, November 14–16, 2023, Denver, CO

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

(1) Collecting the relevant information (from any suitable source),
(2) Maintaining the information – different kinds of information (static/dynamic, qualitative/quantitative, variable/constant, ...), collected from multiple data sources, regarding different components of the system – so that all the data forms one logical structure, and
(3) Providing the information to the user or application in a unified fashion.

There are many sources of system- and hardware-relevant data being in use today — provided by applications, the OS or drivers, or by executing specially tailored benchmarks measuring system properties, to name just a few examples. *sys-sage* does not aim at replacing them; it rather uses the information they provide and simplifies and unifies the way the data from different sources is combined, correlated, and offered to the user. Hence, our core capability is to **(2) maintain and (3) provide the information captured by or via *sys-sage***, while its design allows us to (1) import any relevant information from any existing source. This design enables *sys-sage* to drastically simplify the procedure of integrating and providing HW-relevant data to target applications.

3 INTERNAL DESIGN

Each tool using *sys-sage* defines the information to be uploaded to and to be managed by the library. It is provided by already existing tools and APIs. Once the data is present, the user, which can be an application, a resource manager, a runtime system, or any other program, can query the stored data. In addition, the user can add or modify arbitrary attributes to reflect changes in the system state or new information as it becomes available.

In *sys-sage*, the properties of an HPC system are represented in the form of *Components* and *Data Paths*, which are interlinked with each other, providing the correlation of the different information.

Components have a hierarchical tree structure, providing a construct that is easy to understand for the user and is easy to navigate. It forms the core of *sys-sage*, and all additional information (static or dynamic) is connected to and referenced from it. Each *Component* is of a particular *Component Type* — classes derived from different parts of computer systems so that their specific attributes and functionalities can be represented. Example *Component Types* are `Node`, `Storage`, `Memory`, `Chip`, `Cache`, or `Core`.

Data Paths are a construct that carries information about the relation of two arbitrary *Components*. Each *Data Path* has a source and a target *Component*. Apart from that, no other rules apply — a *Data Path* can store any information. It is up to the user to define what information the *Data Paths* carry.

3.1 Importing Data to *sys-sage*

The information about hardware, which *sys-sage* handles, comes in at different stages of an application life-cycle. Some data is already available before launching the application, some can be queried at the startup, while other information changes dynamically during the runtime and hence needs to be polled repeatedly. The design of *sys-sage* allows querying and storing all this information.

sys-sage imports the data already available before the launch through so-called *Data Sources*, such as hwloc or benchmarks, that provide the needed data. They are read by the so-called *Input Parsers* and transferred to structures recognized by the library's *Internal Representation*. There are *Default Input Parsers* available for the frequently-used *Default Data Sources*, which can be used out-of-the-box. For other *Data Sources*, users can easily write their *Custom Input Parsers* to upload them to *sys-sage*.

The information that is first available when the application is started is stored in *sys-sage* and read through its API. This way, the user of *sys-sage* can add, update, or delete any piece of information stored in the library. *sys-sage*, therefore, also works as a hardware-related information data storage for the application that uses it.

4 USE CASES

We integrate *sys-sage* into multiple scenarios to show its usability as well as to improve the quality of said scenarios.

4.1 Cache-aware Algorithm vs. Dynamically Changing Cache

Specific algorithms (such as stencil computations in our use-case) can profit from ordering the operations so that a single piece of data gets repeatedly reused while in cache. That is achieved by splitting the domain into smaller pieces that fit in the (L3) cache.

However, modern hardware features software options to dynamically restrict access to a fraction of the L3 cache for specific cores or processes to enable fair resource sharing. This dynamic information renders the static L3 size useless; knowing the fraction itself is also insufficient. Only connecting these two pieces of information together can result in correctly estimating the available L3 size.

sys-sage provides the correct value with minimal computational and implementation overhead, enabling the application to keep profiting from tiling even on systems with dynamically changing L3 cache. Using our approach, we achieved a speedup of up to **2.05x** when restricting to 2/11 of L3 over the static tiling approach, adding only ca. 10 lines of code.

4.2 Capturing Memory Access Data in sys-sage

PEBS memory access samples enable collection of detailed information about sampled load operations, such as the load latency or the cache level that served the particular request. This information is collected by the Mitos [3] tool to be further analyzed and visualized by the MemAxes [2] tool. The cache/memory information and the issuing core information enable us to assign the samples to the underlying hardware to allow analyses such as load imbalance.

Since modern hardware has gotten much more complex in the last few years, MemAxes needs to be adapted to reflect the architectural specifics of the new hardware, which is now much more diverse. This is now possible by integrating *sys-sage* as a back-end for storing the HW-related samples. This way, any hardware configuration can be presented and adequately displayed with the respective samples. Moreover, after switching to *sys-sage*, we could open MemAxes to other inputs, such as AMD's IBS samples or simulator-based traces. Hence, *sys-sage* enables larger flexibility in the analyzed HW as well as the input data format while taking the data management burden off from end tools, like MemAxes.

REFERENCES

[1] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. 2010. hwloc: A generic framework for managing hardware affinities in HPC applications. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 180–186.

[2] Alfredo Giménez, Todd Gamblin, Ilir Jusufi, Abhinav Bhatele, Martin Schulz, Peer-Timo Bremer, and Bernd Hamann. 2017. Memaxes: Visualization and analytics for characterizing complex memory performance behaviors. *IEEE transactions on visualization and computer graphics* 24, 7 (2017), 2180–2193.

[3] Alfredo Giménez, Benafsh Husain, David Böhme, Todd Gamblin, and Martin Schulz. 2015. Mitos: A Simple Interface for Complex Hardware Sampling and Attribution.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009