Towards A Programmable Analysis and Visualization Framework for Interactive Performance Analytics

Tanzima Z. Islam
tanzima@txstate.edu
Texas State University

Alexis Ayala, Quentin Jensen
ayalaa2,jensenq@wwu.edu
Western Washington University

Khaled Ibrahim
kzibrahim@lbl.gov
Lawrence Berkeley National Laboratory

1 Abstract

Understanding the performance characteristics of applications in modern HPC environments is becoming more challenging due to the increase in the architectural and programming complexities. HPC software developers rely on sources such as hardware counters and event traces to infer performance problems while focusing on designing mitigation strategies. A large number of in-house tools exist in the community, which indicate replicated effort. This paper presents a customizable framework for analyzing performance measurements and visualizing through a web-based interactive dashboard for interactively exploring a large volume of hierarchical information. In this paper, we analyze three ECP applications as use cases and identify as well as optimize problematic resource utilization behaviors exposed by our visualizations. This framework is a step towards a unified platform for visual identification of performance scaling bottlenecks to ease the collaboration between application developers, performance analysts, and hardware vendors.

2 Introduction

Modern systems include architectures with hundreds of components (e.g., L1, L2, memory), and their complex interactions with applications cause performance loss. These machines can log thousands of on-core and off-core events that can explain how an application interacts with the underlying system. Analyzing these events can expose potential optimization opportunities for an application. For example, a large number of page faults may cause a performance issue. Visual inspection of the collected data can help expose apparent patterns that can be useful for designing targeted analysis and optimization strategies.

While this swell of information is undoubtedly useful, this may leave performance analysts with overwhelming information to investigate. A typical performance analysis workflow includes: (1) identifying regions of interest to investigate, (2) identifying how an application utilizes the underlying hardware resources such as cache hierarchy and compute units, (3) pinpointing actual events recorded by the hardware to gain insight into performance problems, and (4) comparing performance across workloads, applications, and configurations (e.g., varying numbers of workers). Frequently, application developers team up with performance analysts to develop in-house tools written in languages such as R, Python to wrangle the vast amount of data collected to gain insights into performance problems quickly. The focus typically stays on devising optimization strategies afterward to mitigate the problems. Since the goal of the in-house analysis tools is often quick data exploration to guide optimization, these tools usually are not designed to be extendable by others. This phenomenon leads to duplicated efforts of developing similar if not the same analysis and visualization pipelines across the HPC community.

To fill this gap, we have developed an HPC performance analysis and visualization framework, Dashing, based on the principles of programmability, extendability, and interpretability of results in mind. The modular design of this Python framework enables plugging in any analysis and visualization techniques of choice. Specifically, in this paper, we present a web-based visualization dashboard for interactive exploration of hierarchical performance data. These interactive visualizations provide both coarse- and fine-grained information about the causes of a target performance metric (e.g., efficiency loss), and a quantitative ranking of these reasons so that a user can prioritize their mitigation strategies. We envision that some of these causes can be easily fixed by the application developers (i.e. domain scientists), while others may need further engagement between application developers, performance analysts with system expertise, and vendors.

Although the types of analysis and visualizations that can be supported by Dashing are practically unlimited, we demonstrate the effectiveness of this framework by addressing the following typical performance analysis questions: (1) explain the impact of resource utilization behaviors on performance, (2) identify individual hardware events that can explain scalability loss, and (3) compare two performance profiles quantitatively across applications. Here, the term resource stands for an arbitrary user-defined group of hardware performance counters. As case studies, we analyze three ECP applications – Nyx3D, IAMR, and INCFLO. Nyx3D is a compressible cosmological hydrodynamics application. IAMR and INCFLO solve incompressible Navier-Stokes equations, which are used by combustion simulations. The INCFLO application is a much smaller proxy application, developed to mimic the resource utilization behaviors of IAMR. For analysis, we extended our previous machine learning techniques [25] to provide counter importance along with resource significance in explaining the parallel efficiency loss as applications scale on a node.

While most works in the literature have focused on developing novel analysis techniques, only recently, the community has recognized the value of visualizations to aid in exploring the plethora of collected information. Several works have developed specific visualizations for specific problem

domains [4, 10, 12, 14, 27]. In contrast, we aim to develop a common platform for all researchers to easily incorporate their methodologies to serve the performance analysis needs of the community.

The contributions of this paper are:

- A novel hierarchical visualization of how the hardware utilization behaviors of applications impact its performance
- An easily extendable and modular framework, DASHING, that implements several visualization techniques both novel and widely used by the performance analysis community
- Optimization opportunities exposed by the DASHING framework for three applications important for the Exascale Computing Project (ECP)

The most notable contribution of this work is that our visualizations identified TLB misses being important contributors to the efficiency loss of Nyx3D and IAMR. We mitigated those issues by loading huge pages of 4MB on Cori. Some performance problems observed due to the use of atomic instructions may need more involved code refactoring, algorithm change or discussion with processor vendor. With the DASHING framework, we envision creating such engagement opportunities for co-design among application developers, performance analysts, and hardware vendors.

We organize the rest of the paper as follows. Section 3 presents a high-level description of the analysis methodologies used in this paper and Section 4 presents the overall workflow of the DASHING framework. Sections 5 and 6 present the experimental setup and the observations from analyzing three ECP applications. Finally, Section 8 summarizes the paper and indicates future research planned in this direction.

3 Background

In this paper, we leverage two methodologies for calculating how significantly hardware components (e.g., cache, memory) impact the performance of applications and how well two performance profiles match. Such performance profiles can be collected by varying applications, workloads, or regions within an application. We extend the analysis techniques from our previous work [25, 26]. This section presents these two methodologies from a high-level.

Identifying important hardware events can be formulated as a feature selection problem in machine learning, where a number of features (e.g., cache miss, resource stalls) can explain the performance problems (e.g., increase in execution time) of applications. The following sections explain the target and the features used for our analysis.

Features: Hardware Counters and Resources On-node performance is typically constrained by hardware bottlenecks on the processors or in the memory system. Hence, we use hardware performance counters to explain performance loss. Hardware counters are registers in modern architectures that keep track of events such as cache misses, which impact the overall performance of applications. However, there are hundreds of such events that can be collected via PAPI [22]. To avoid overwhelming users with fine-grained information such as the count of each hardware event, we categorize the events to high-level groups based on the hardware components they affect. We refer to these high-level groups as "resources".

Target: Performance loss This paper focuses on explaining the performance loss of applications as they strongly scale on a node based on their hardware component utilization behaviors. Strong scaling is defined as increasing the number of workers (e.g., threads, processes) while keeping the total problem size fixed. In this paper, we define performance loss as the loss of efficiency as an application scales (Equation 2), since this metric impacts the final time to solution. Efficiency loss typically increases as an application uses more cores on a multi-core machine. Since every event (e.g, cache miss) measured by a hardware performance counter has a non-zero penalty, our goal is to identify those events that follow the same scaling pattern as the efficiency loss.

$$\text{efficiency\_loss}(p) = 1 - \frac{T(1)}{(T(p) \times p)} \tag{1}$$

where p = number of workers (threads or processes) and $T(p)$ = execution time when using p workers.

Resource Significance Measure (RSM) This paper leverages the machine learning approaches published in our previous work [25, 26]. We compute the impact of resource utilization on efficiency loss by building a linear predictive model with the collected hardware performance counters. In particular, we leverage the extended Orthogonal Matching Pursuit (eOMP) algorithm presented in [25]. The eOMP algorithm builds on the traditional OMP algorithm presented in [11], which selects the smallest and diverse subset of counters to describe the performance loss.

However, hardware performance counters from different hardware resources can be inter-correlated, which will not be selected by a traditional OMP algorithm. Instead, the eOMP algorithm selects τ most correlated counters (instead of one) and creates a discrete distribution based on their correlations, i.e., the higher the correlation, the higher probability to be chosen in that step. It then randomly picks a counter using the discrete distribution, computes the residual, and repeats this process until a desired sparsity (user defined) is met or the model achieves sufficient fidelity. We repeat this randomized algorithm 5000 times independently and obtain the final coefficients for counters by averaging the solutions in the ensemble.

The algorithm then computes a belief metric for each of the counters, if they have a non-zero coefficient in the sparse representation. The belief value is inversely proportional to the reconstruction error of each counter. The final step combines counter-wise beliefs to their corresponding resources by obtaining the compound evidence [9].

Similarity Analysis We leverage our previous work [25, 26] to compare the resource utilization behaviors of a pair of applications. The comparative analysis approach computes a distance between the two feature spaces defined by the set of performance counters. From high-level, the methodology computes (1) the low-dimensional projections (PCA)

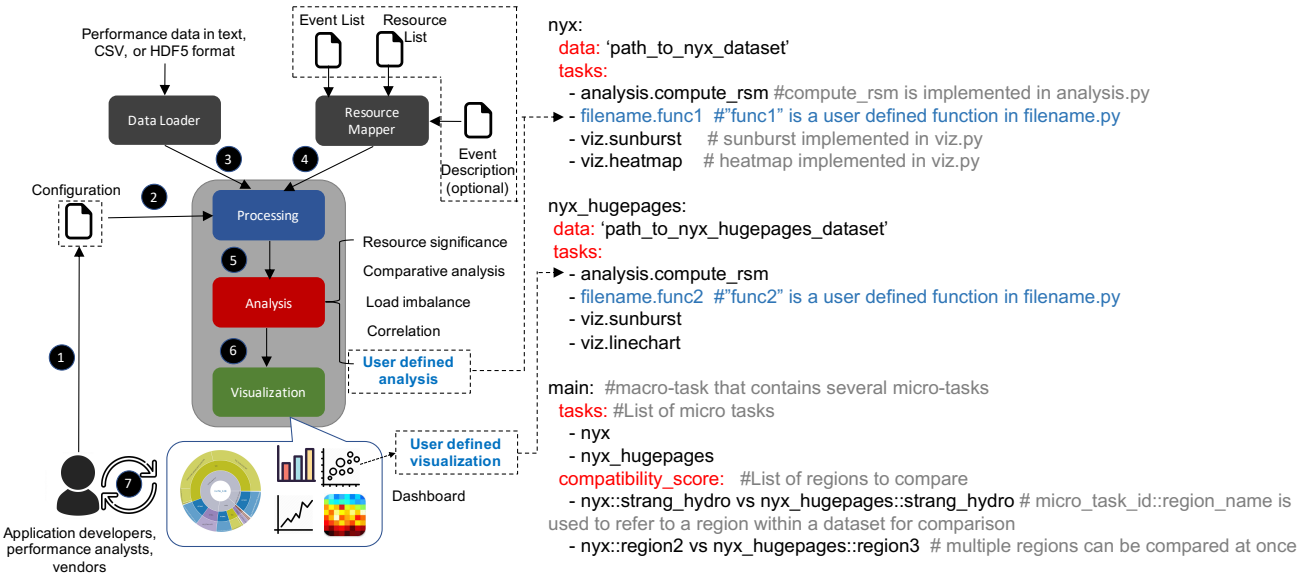


Figure 1: (a) Workflow of the Dashing framework. The dotted boxes represent points of interaction between users and the framework for leveraging Dashing for other use cases (b) Example configuration file with hierarchical structure.

of the performance subspaces (d principle directions), (2) measures the principle angles between each of the principle directions (θ_i along direction i), and (3) the distance between the distributions (λ_i along direction i) of the collected performance data. Then the compatibility score of a resource (r) is computed as:

$$\text{compatibility\_score}(r) = 1 - \frac{1}{d}\sum_i \theta_i \lambda_i \qquad (2)$$

4 Approach & Framework

This paper focuses on presenting visualizations for analyzing application performance in a modular framework—Dashing—that is easy to extend. The visualizations are interactive and can guide users to (1) correlate the utilization of the hardware components to the performance loss of an application as it scales, (2) identify hardware events that can explain efficiency loss, and (3) enable quantitative comparison of the performances between two applications. In the context of performance analysis, we refer both application developers (i.e., domain scientists) and performance analysts with system expertise as users. Figure 1 presents the overall workflow of the Dashing framework, which mainly performs three tasks: (1) Processing (2) Analysis and (3) Visualization. This section describes the components that implement these tasks and the end-to-end use of the framework in details.

4.1 Processing

Data Loader: The processing task starts with reading data from the performance files collected from various measurement tools and converting those into a Python dictionary. Performance measurement tools write data in different formats such as HDF5, Comma Separated Values (CSV), and plaintext. Currently, the `data_loader` module supports files written in the aforementioned formats. To read files in user-defined format, a performance analyst needs to write a Python function to read data to a Python dictionary and return that dictionary.

Resource Mapper: The `resource_mapper` module reads the event-to-group mapping from a user provided CSV file or calls a user-defined function to map hardware events (or features) to resources (groups). In this work, we leverage the event-to-resource mapping function developed from our previous work [25] to categorize the hardware events on Cori. Table 1 presents the resource groups used in this work for the Intel Haswell architecture.

4.2 Analysis

The `analysis` module essentially is a wrapper that iterates through a list of analysis functions written in Python. This function pointer based method enables users to perform any analysis of their choice, as long as the return value is a Python dictionary of structure $< key1, key2, key3 > \rightarrow value$. In this paper, the tuple represents $< region, resource, event > \rightarrow importance$. The two analysis methodologies used in this work address the following research questions: (1) which regions in an application constitute the largest run time, (2) which hardware resource utilization can explain the efficiency loss curve as an application scales on a node, (3) which resource utilization behaviors match between two applications. For addressing the first question, the Dashing framework uses execution times to rank regions of interest. For addressing the second and the third questions, the Dashing framework builds on the machine learning techniques presented in our previous work [25, 26]. A high-level overview of the methods is presented in Section 3.

4.3 Visualization

The `visualization` module is responsible for visually representing the output from different analysis into a web-based dashboard. The goal of these visualizations is to present hierarchical information in an interactive manner. The DASHING framework provides detailed meaning of each performance event along with any known optimization techniques as a tool tip text for each event. This description of each event, provided through the `configuration` file, contains $< eventname, description >$ pairs for each event. Since the tool will be made available on Github to the community, we envision both system analysts and architects can per take in enhancing this tool by annotating the event description files, written in the CSV format.

With the goal of easy interpretability, we have created several visualizations corresponding to several commonly performed analysis tasks. This list of visualizations present information from a coarse to fine-grained level:

4.3.1 **Heat map (Figures 4a, 4b)**compares the RSM scores of resources per region, giving a broad overview about which resources are heavily used by which regions. This summary visualization provides a quick way for comparing the resource utilization behaviors across all annotated regions while abstracting away a lot of details.

4.3.2 **Sunburst (Figure 2)**presents a large amount of hierarchical information in a visually intuitive manner. Each section of each circle represents its relative cost to its parent. Each section is clickable, expanding that section to fill the whole chart (Figures 2b, 2c). A single sunburst chart represents an application. Moving outwards, the next circle represents the contribution of each region to the overall execution time of the application. Then, the importance of each resource that contributes to explaining the efficiency loss of its parent region. Finally, the importance of each hardware counter that contributes to its respective resource. In this fashion, a user can quickly identify what regions of code are high-cost, and then zoom in on why. This coarse-to-fine grained visualization is powerful in its ability to help a user discover specific ways to optimize their code. The tool shows only significant events attributed to efficiency loss, down-selecting from hundreds of system events.

4.3.3 **Interactive line chart (Figure 3)**shows the values of hardware counters normalized within a region, as an application scales. Each line is colored by its resource, shown alongside the region's efficiency loss curve (bold black). Counters that follow the efficiency loss curve tightly are likely to explain why the efficiency loss occurred. This visualization enables users to investigate the trends in events compared to the trend in the target variable (e.g., efficiency loss).

4.3.4 **Scatter plot (Figure 4d):**showcases the comparison between two performance profiles. The X axis shows the RSM scores for the first application, and the Y axis shows how well the second application mimics the resource utilization behaviors of the first one. A lower value of compatibility indicates greater dissimilarity between the two performance profiles. This visualization was proposed in our previous work [25] and has been integrated in the DASHING framework. Although, the Sunburst chart can present a comparative view of the resource utilization across regions, it is more suitable for a hierarchical view of the performance characteristics of a single region. In contrast, the heat map is more suitable for a summarized view of the resource utilization behaviors across regions. Hence, the DASHING framework includes both.

Resource Group on Intel Haswell	**ID**
Prefetcher	PRE
Floating point unit	FP
Branch instruction and prediction units	BR
L1 cache	L1
L2 cache	L2
L3 cache	L3
Translation look aside buffer	TLB
Offcore events	OFF
Micro operations	UOPS
Front-end	FE
Non-floating point arithmetic unit	ARITH
Atomic events	ATOMIC
Translation look aside buffer	TLB
Memory	MEM
Events to prune	UNDEFINED

Table 1: Resource groups on Cori

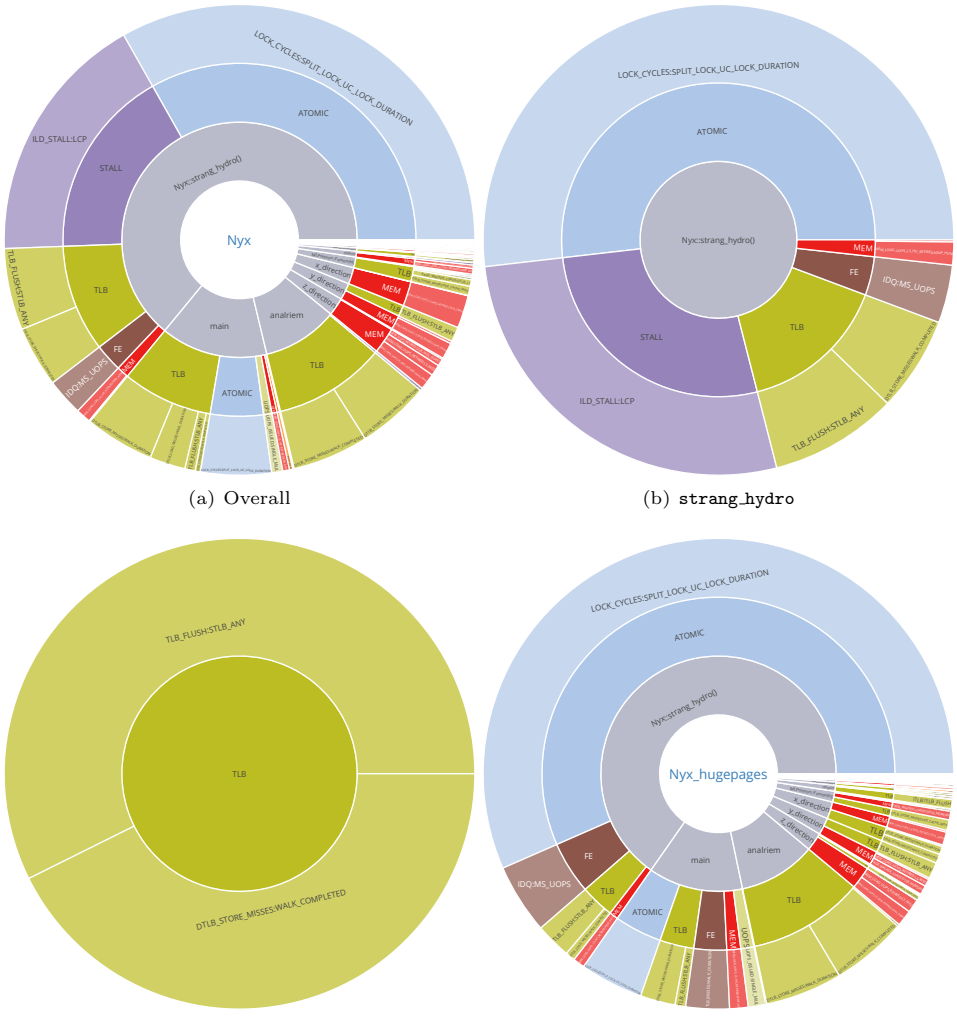
4.4 Configuration

DASHING provides a simple configuration file-based approach to execute and extend DASHING. The configuration file is written in YAML. The configuration file is organized as a hierarchy of tasks. We assume that performance data is collected for code regions (e.g., function) and is organized under a name (region name). This organization enables several regions in the same application to be collected and analyzed at once. It is written in Python 3.7, hence programmable. The command line invocation of DASHING is: `python driver.py config.yaml`.

Figure 1b shows an example configuration file for generating Figures 2 and 3. The fields in red (`data` and `tasks`) describe keywords, and the items under the `tasks` are functions to apply on the data. To extend DASHING, users can enlist their Python function names under `tasks` (e.g., `func1`, `func2`). If the task pertains to a specific dataset, then it should be listed under a micro task (e.g., `compute_rsm` on Nyx3D data). The `main` task provides access to all datasets (i.e., micro tasks). Hence, tasks to analyze across several datasets should be enlisted under `main`. For e.g., the `compatibility_score` is computed between two regions in two datasets. These regions are accessed by specifying their micro task names, a "::" operator, and the region names after that (e.g., `nyx::strang_hydro` where Nyx3D is the micro task name and `strang_hydro` is the region name). In the future, we will make the configuration editing capability available via the web-based dashboard itself to reduce the adoption barrier to application developers and increase their productivity.

5 Experimental Setup

In this section, we describe the system and applications we use to demonstrate the use cases of the DASHING framework. For all applications, we used the sparsity parameter to be 3 (as was found to be sparse enough in our previous work [25]).



(a) Overall

(b) `strang_hydro`

(c) TLB in `strang_hydro`

(d) With hugepages=4M

Figure 2: Interactive graph for Nyx3D shows that ATOMIC, TLB, and Instruction stalls are the three major groups to optimize for the most expensive region, `strang_hydro`.

5.1 System Description

We ran our experiments on Cori [19], a Cray XC40 supercomputer with 29.5 Petaflop/s peak performance. Cori consists of 2,388 Intel Xeon E5-2698 (Haswell) and 9,688 Intel Xeon Phi 7200 (Knights Landing) processors. Each Haswell compute node has two sockets, each socket has 16 cores with 36.8 Gflops/core and 128 GB memory. Each core has its own L1 and L2 caches, with 64 KB (32 KB instruction cache, 32 KB data) and 256 KB, respectively; there is also a 40-MB shared L3 cache per socket. In this paper, we present experimental results on the Intel Haswell nodes only. Table 1 presents the hardware resources on Cori. We collected more than 200 hardware counters and categorized them based on the open source PMU tools project [1] for Intel CPUs. We leave performance analysis on the Intel Xeon Phi processors for future work.

5.2 Applications

Nyx3D is an adaptive mesh, compressible cosmological hydrodynamics simulation code that leverages the AMReX framework. The application is written using a combination of Fortran 2003 and C++11 with parallelism supported by a hybrid MPI+OpenMP model [20]. Nyx3D solves equations of compressible hydrodynamics on an adaptive grid hierarchy coupled with an N-body treatment of dark matter.

IAMR is a parallel, adaptive mesh refinement (AMR) code that solves the variable-density incompressible Navier-Stokes equations. The IAMR source code can be found at [17].

INCFLO [18] is a proxy application similar in purpose to IAMR, but has a simplified code base. INCFLO has been

developed to only capture the computation and the performance characteristics of the advection, diffusion, and projection operations for solving the incompressible Navier-Stokes equations (implemented in the `advance` method) in its much larger counterpart, IAMR.

6 Results

In this section, we present the effectiveness of DASHING visualizations using three ECP applications. We collected the hardware performance counters for the three applications Nyx3D, IAMR, and INCFLO using the Perf-dump [21] tool. Due to page limit, we only present a subset of these visualizations to demonstrate the usability of the developed framework. Specifically, we present the following visualizations for answering common questions asked during a typical performance analysis phase:

- Heat map: Comparison of the resource utilization behaviors across regions
- Sunburst: Comparison of the execution times, resource importance, and contributing hardware events as well as their importance
- Interactive line chart: Comparison of the features (e.g., hardware performance counters) with the target (e.g., execution time, efficiency loss) across various configurations (e.g., varying number of workloads, workers)
- Scatter plot: Comparison of resource utilization between two performance profiles. These profiles can be collected by varying workloads of the same application, or different regions, or even different applications.

The following section presents specific examples on how these visualizations can help both application developers and vendors identify performance optimization opportunities in applications.

6.1 Nyx3D

In this section, we present the resource utilization behaviors of 19 annotated regions in Nyx3D. We identified these regions through hot spot analysis using the Craypat-lite [15] tool. The `main` region represents the rest of the methods in the Nyx3D application that we did not explicitly annotate (i.e., not the `main` method itself).

The Sunburst visualization provides an interactive way to present a large amount of information, such as execution times of the regions, resource and counter importance for all regions in a concise manner. Figure 2a shows the breakdown of the overall performance of Nyx3D, 2b filters by a region, and 2c filters by a resource. Tool tip text (not shown in figure) shows the numeric importance values and execution times of the regions. Section 4.3.2 explains how to interpret the information at each level of the sunburst chart. All regions, resources, and events are sorted based on their values (e.g., execution times, importance) and organized in counter-clockwise manner.

The observations that can be made from Figure 2 are:

- `strang_hydro` explains 65% of the overall execution time, hence the most important region to optimize. The specific value (65%) is presented as a tool tip text when users hover over that region (not shown in figure).
- TLB is the most important resource with a significance score of 43%. This indicates that all or a subset of the TLB counters together can explain 43% of the efficiency loss curve as Nyx3D scales on a node. Hardware counters typically count hardware events, where the increase of an event means cycles spent.
- Figure 2c shows that DTLB (Data TLB) misses during store operations result in a large number of walks through all page sizes. A user can easily optimize this performance problem by using huge pages for Nyx3D on Cori.
- Figure 2d shows that using 4M huge pages on Cori indeed reduces the impact of TLB misses on the performance of Nyx3D (reduced from 11% to 3%). The information presented by our analysis and visualizations enable a user to identify potential opportunities for optimization.
- Figures 2a and 2d also show that the use of ATOMIC instructions contribute largely to the increase in efficiency loss for Nyx3D. Optimizing the ATOMIC operations may not be an easy step for an application developer alone. This observation presents an opportunity for engaging with the vendors to discuss potential optimizations.

Figure 3 shows a comparison of the normalized values of the hardware performance counters and the efficiency loss of each region as the application scales. The bold black line indicates the efficiency loss of the region as it scales on more workers, and the individual lines correspond to each important event. Hardware events are filtered by their importance to declutter the visualizations. Users can set the threshold to an arbitrary low value (e.g., zero) to control the density of information.

6.2 INCFLO

We ran INCFLO with multiple input sizes. The number of cells parameter was varied from $32^3(small)$, $64^3(medium)$, and $128^3(large)$. Due to the page limit, we only present a subset of the visualizations. These results clearly expose potential opportunities for optimizing the performance of the INCFLO application across multiple regions.

The heat map visualizations identify a pattern of resource utilization behaviors across all annotated regions of the INCFLO application. Figures 4a and 4b show the resource utilization behaviors across three regions annotated within the INCFLO application for the medium and large problem sizes, respectively. From Figure 4a and 4b, we can observe that while TLB is an important resource across all the regions for the small input size, ATOMIC is important for the large one. This summary information is further augmented by Figure 4c, which shows a detailed report of the regions, resources, and events for the large workload. From Figure 4c, we can observe that a large number of TLB store misses occurred, which contributed to the TLB resource impacting the efficiency loss of the INCFLO application. An application developer can reduce such TLB misses by increasing the TLB page size, which on Cori can be done by loading a module called "cray-hugepages4M". The default size of the hugepages on Cori is 2M.

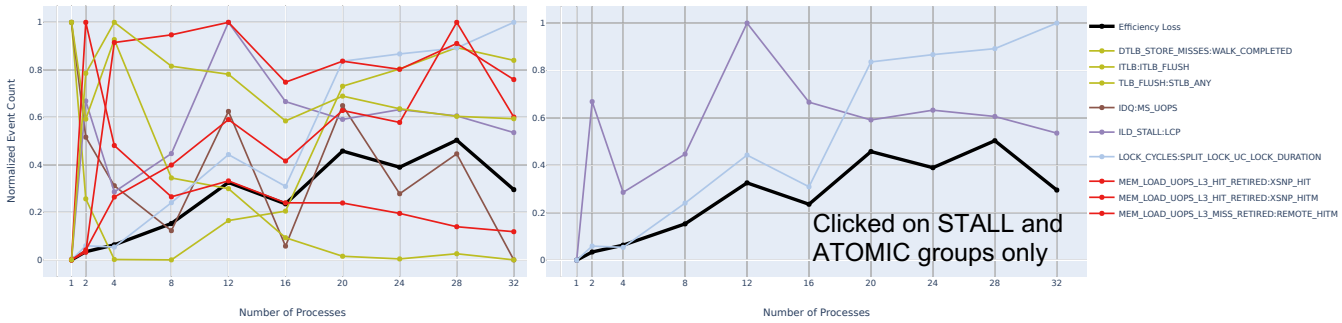
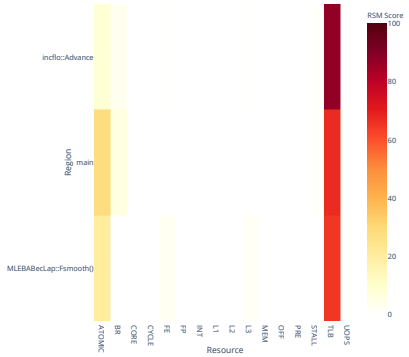
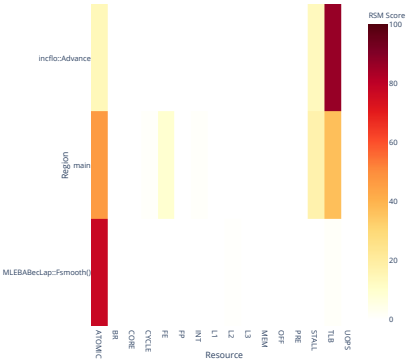


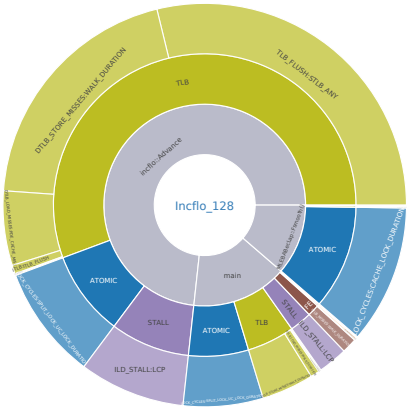
Figure 3: Interactive line chart shows the comparison of the normalized values of the features (e.g., hardware performance counters) to a target variable (e.g., the efficiency loss). Clicking on an event in the legend toggles a group.



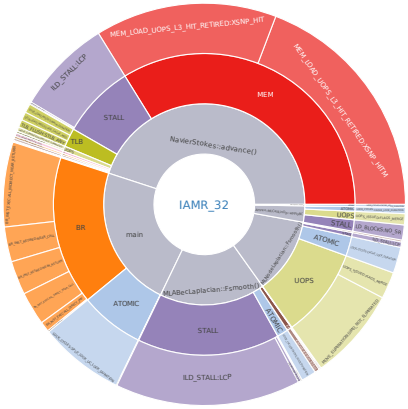
(a) Incflo with $n\_cell = 64^3$



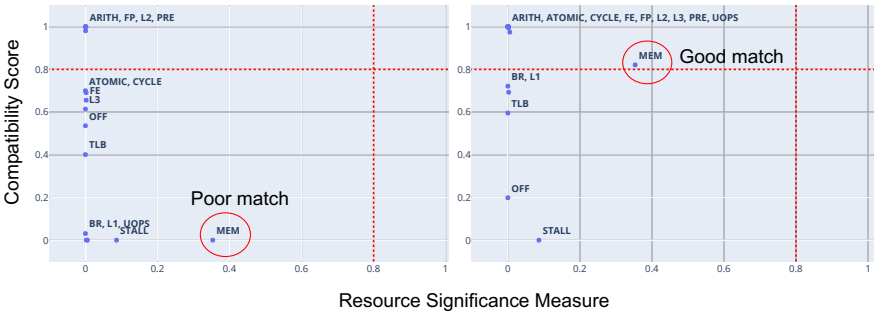
(b) Incflo with $n\_cell = 128^3$



(c) Overall performance characteristics for INCFLO with $n\_cell = 128^3$



(d) Overall performance characteristics for IAMR



(e) Comparison between IAMR and INCFLO $n\_cell = 32^3$ and 128^3

Figure 4: (a) and (b) Comparison of resource utilization behaviors across regions. (c) Overall performance shows TLB is the most important resource. Application developers may use the hugepages to improve the performance of INCFLO when using a large workload. (d) MEM is the most important resource for IAMR. (e) Comparison between IAMR and INCFLO (with two workloads (small and large).

6.3 IAMR

The DASHING framework similarly analyzes and visualizes the performance characteristics of the IAMR application. Due to page limit, we only present the comparative analysis results between IAMR ($n\_cell = 32^3$) and INCFLO ($n\_cell = 32^3$, 64^3, and 128^3). The INCFLO application has been developed to capture the advection, diffusion, and projection tasks for solving the incompressible Navier-Stokes equations. In this section, we only present a comparison between the performance characteristics of the `advance` methods (implementing all three of the tasks mentioned above) in both of these applications. Figure 4d shows the comparison between IAMR with workload 32^3 and INCFLO with workloads 32^3 and 128^3. The X axis shows the resource significance measure for IAMR. Higher values mean a greater importance to cover. The Y axis shows the compatibility score; higher values mean a more similar behavior.

From Figure 4e we can observe that MEM is the most important resource for IAMR, however INCFLO with the small input size (Figure 4e-left) does not cover the behavior well. In this work, we deem a compatibility score of 0.8 or higher as a good match (shown using red dotted horizontal line). On the other hand, the MEM utilization behavior for INCFLO with the large input size (Figure 4e-right) covers that of IAMR well. This surprising observation indicates that further investigation is needed to ensure that INCFLO mimics IAMR.

7 Related Work

ParaProf [5], Tau [24], YAViT [2], HPCToolKit [3] are widely used HPC tools that provide integrated analysis and supporting visualizations for identifying performance bottleneck in applications. While these are massive tools with a large user base, they often lack the flexibility of a simple Python framework for interactive exploration of the collected data. Visualizations of supercomputer networks [12], network traffic [8], scalability trend [4], job placements [7], power consumption [16], flops visualization for threads [10], communication [23] are great examples of targeted visualizations that have been developed by the community. All of these works focus on developing intuitive visualizations for presenting data for a specific problem. In contrast, we envision that this work will lead to the development of a unified programmable framework for all HPC performance analysis and visualization needs.

PaScal [4] is a visualization tool for studying scalability of applications, a goal similar to our own. Performance monitoring and visualization dashboard in [13] focuses on a system for collecting cluster health statistics and visualizing a large volume of data. This system comes equipped with a large number of components that enable real time monitoring, data collection in addition to visualization. Furthermore, there are works on visualizing calling contexts [6], memory performance behaviors [14], and frameworks for projecting performance data over simulation geometry [27]. While the visualizations are useful, none of the work aims at providing a unified framework for all analysis and visualization needs of the HPC performance community. In that regard, these analysis and visualization techniques are complementary to our work.

8 Conclusion

This paper presents a novel visualization framework for presenting hierarchical information about application performance and identifying critical resources limiting scaling efficiency. Our framework introduces an interactive web-based dashboard that implements several performance visualization methods that can be useful for application developers, performance analysts, and system vendors alike.

Although DASHING is written in Python, the visualization dashboard can be integrated in a Cloud instance for providing interactive exploration of the performance dataset. Such a tool can enable collaboration across multiple research labs, academics, and vendors to share data without overwhelming users with raw data. The DASHING framework supports programmability through a configuration file, extendability through function pointers, and interpretability through interactive visualizations, for which we plan to provide on-the-fly customization in the future.

In the future, we will incorporate information from compiler-assisted annotations and compiler generated optimization reports to present a unified visual that is easy to interpret and take actions on. We will also continue to add more visualizations to the DASHING framework, so that one can fully configure the axes from the configuration file to invoke any of the functions. This work is the first step towards providing a framework for visual performance analytics in a comprehensive manner. We would also like to support visualization for correlations across the performance counters in a meaningful way so that the ones that are always correlated can be identified. This list can be used to filter the number of hardware performance counters that need to be collected. We demonstrate the use of DASHING in the analysis and tuning of Nyx3D, IAMR, and INCFLO.

References

[1] [n. d.]. PMU-Tools: A Collection of Tools for Profile Collection and Performance Analysis on Intel CPUs on top of Linux Perf. https://github.com/andikleen/pmu-tools.

[2] Omar Aaziz, Ujjwal Panthi, and Jonathan Cook. 2017. YAViT (Yet Another Viz Tool): Raising the Level of Abstraction in End-User HPC Interactions. In 2017 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 814–817.

[3] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs. Concurrency and Computation: Practice and Experience 22, 6 (2010), 685–701.

[4] F de A Alex and Samuel Xavier-de Souza. [n. d.]. PaScal Viewer: A Tool for the Visualization of Parallel Scalability Trends. In Programming and Performance Visualization Tools: International Workshops, ESPT 2017 and VPA 2017, Denver, CO, USA, November 12 and 17, 2017, and ESPT 2018 and VPA 2018, Dallas, TX, USA, November 16 and 11, 2018, Revised Selected Papers. Springer, 250.

[5] Robert Bell, Allen D Malony, and Sameer Shende. 2003. Paraprof: A portable, extensible, and scalable tool for parallel performance profile analysis. In European Conference on Parallel Processing. Springer, 17–26.

[6] Alexandre Bergel, Abhinav Bhatele, David Boehme, Patrick Gralka, Kevin Griffin, Marc-André Hermanns, Dušan Okanović,

Olga Pearce, and Tom Vierjahn. 2017. Visual Analytics Challenges in Analyzing Calling Context Trees. In Programming and Performance Visualization Tools. Springer, 233–249.

[7] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. 2013. There goes the neighborhood: performance degradation due to nearby jobs. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. ACM, 41.

[8] Harsh Bhatia, Nikhil Jain, Abhinav Bhatele, Yarden Livnat, Jens Domke, Valerio Pascucci, and P-T Bremer. 2018. Interactive Investigation of Traffic Congestion on Fat-Tree Networks Using TreeScope. In Computer Graphics Forum, Vol. 37. Wiley Online Library, 561–572.

[9] Isabelle Bloch. 1996. Some aspects of Dempster-Shafer evidence theory for classification of multi-modality medical images taking partial volume effect into account. Pattern Recognition Letters 17, 8 (1996), 905–919.

[10] Ronny Brendel, Bert Wesarg, Ronny Tschüter, Matthias Weber, Thomas Ilsche, and Sebastian Oeste. 2017. Generic library interception for improved performance measurement and insight. In Programming and Performance Visualization Tools. Springer, 21–37.

[11] Alfred M Bruckstein, Michael Elad, and Michael Zibulevsky. 2008. On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations. IEEE Transactions on Information Theory 54, 11 (2008), 4813–4820.

[12] Shenghui Cheng, Wen Zhong, Katherine E Isaacs, and Klaus Mueller. 2018. Visualizing the topology and data traffic of multidimensional torus interconnect networks. IEEE Access 6 (2018), 57191–57204.

[13] Tommy Dang. 2017. Visualizing Multidimensional Health Status of Data Centers. In Programming and Performance Visualization Tools. Springer, 273–283.

[14] 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.

[15] Cray Inc. 2016. Craypat-lite: Cray Performance Measurement and Analysis Toolset. https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide/craypat-lite.

[16] Stephanie Labasan, Matthew Larsen, Hank Childs, and Barry Rountree. 2017. PaViz: A Power-Adaptive Framework for Optimizing Visualization Performance.. In EGPGV. 1–10.

[17] Lawrence Berkeley National Laboratory. [n. d.]. IAMR: A Parallel, Adaptive Mesh Refinement (AMR) Code that Solves the Variable-density Incompressible Navier-Stokes Equations. https://github.com/AMReX-Codes/IAMR.

[18] Lawrence Berkeley National Laboratory. [n. d.]. Incflo: An AmrCore-based Code that Solves the Incompressible Navier-Stokes Equations. https://github.com/AMReX-Codes/incflo.

[19] Lawrence Berkeley National Laboratory. [n. d.]. NERSC's Cray XC40 Supercomputer. https://www.nersc.gov/users/computational-systems/cori/.

[20] Lawrence Berkeley National Laboratory. [n. d.]. Nyx: An Adaptive Mesh, Massively-parallel, Cosmological Simulation Code. https://github.com/AMReX-Astro/Nyx.

[21] Lawrence Livermore National Laboratory. [n. d.]. Perf-dump: A tool for collecting PAPI counter values per-process, per-thread, and per-timestep. https://github.com/scalability-llnl/perf-dump.

[22] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. 1999. PAPI: A portable interface to hardware performance counters. In Proceedings of the department of defense HPCMP users group conference, Vol. 710.

[23] Philip C Roth. 2017. Improved accuracy for automated communication pattern characterization using communication graphs and aggressive search space pruning. In Programming and Performance Visualization Tools. Springer, 38–55.

[24] Sameer S Shende and Allen D Malony. 2006. The TAU parallel performance system. The International Journal of High Performance Computing Applications 20, 2 (2006), 287–311.

[25] Jayaraman J. Thiagarajan Tanzima Z. Islam. 2013. A Machine Learning Framework for Performance Coverage Analysis for Proxy Applications. IEEE transactions on signal processing 61, 7 (2013), 1644–1656.

[26] Jayaraman J. Thiagarajan, Rushil Anirudh, Baibhab Kailkhura, Nikhil Jain, Tanzima Z. Islam, Abhinav Bhatele, Jae-sung Yeom, and Todd Gamblin. 2018. PADDLE: Performance Analysis Using a Data-Driven Learning Environment. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 784–793. https://doi.org/10.1109/IPDPS.2018.00088

[27] Chad Wood, Matthew Larsen, Alfredo Gimenez, Kevin Huck, Cyrus Harrison, Todd Gamblin, and Allen Malony. 2017. Projecting performance data over simulation geometry using SOSflow and ALPINE. In Programming and Performance Visualization Tools. Springer, 201–218.