VISILIENCE: An Interactive Visualization Framework for Resilience Analysis Using Control-Flow Graph

Hailong Jiang [1] Shaolun Ruan [2] Yong Wang [2] Bo Fang [3]
Qiang Guan [1]

[1]Kent State University [2]Singapore Management University [3]Pacific Northwest National Laboratory

Outline

Background: Soft Errors in High-Performance Computing (HPC) system

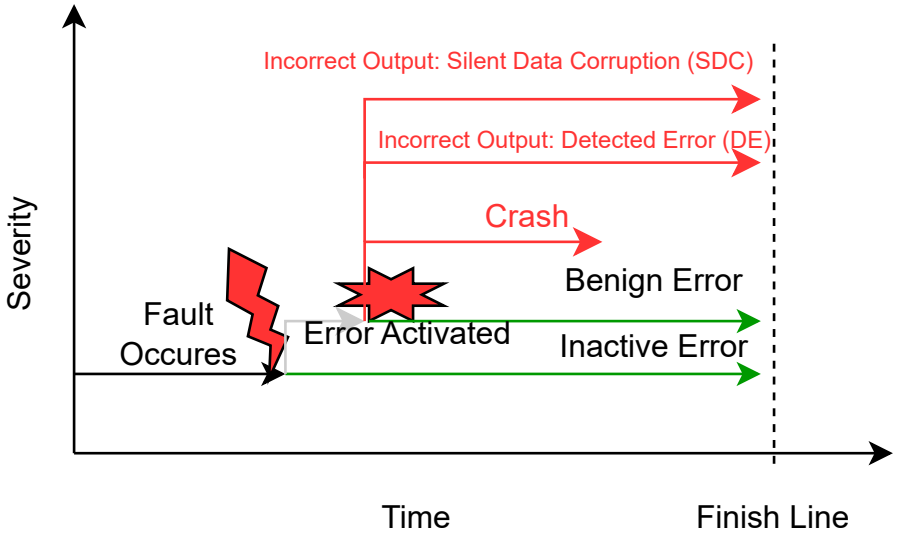

Figure: Types of failures [1]

- **Fault**: Root cause of an error, usually physical defects or software bugs
- **Benign**: The program output matches that of the error-free execution even though a fault occurred during its execution
- **Silent Data Corruption (SDC)**: The program output does not match, it is called **Silent Data Corruption (SDC)**.
- **Crash**: The OS terminates the program due to the error.

Background: Resilience Analysis Methods

Y-Branch:

Y-Branch: To characterize outcome-tolerant branch (Y-branch)instances. The Y-Branch model uses values “0" and “1" to represent Y-Branch and non-Y-Branch, respectively.

IPAS:

IPAS take the instruction features as input and outputs the class of this instruction: *Class 0:* non-SOC-generating instruction or *Class 1*: errors SOC-(**S**ilent **O**utput **C**orruption)-generating instruction.
We use the rate of SOC-generating instructions in this basic block to represent the resilience of it.

TRIDENT:

TRIDENT [2] estimates the SDC probability of individual instruction and the entire program without performing any FIs. We use it to calculate SDC probabilities of dynamic instructions and use SDC probabilities to represent the resilience.

Motivations

- Instruction level information is hard to follow.

  For example, IPAS identifies the SOC-generating instructions but does not explicitly help users scope the instructions;

- The resulting resilience characteristics of a series of program states can be scattered, lacking a holistic view for the users.

  For example, TRIDENT *models the SDC probability of state transitions where the state dependency is not presented in the final output.*

- To classify and summarize the unstructured resilience-related data generated by those approaches requires enormous efforts for large-scale HPC applications;

- There is no platform to post-analyze the results from different resilience frameworks.

VISILIENCE Overview

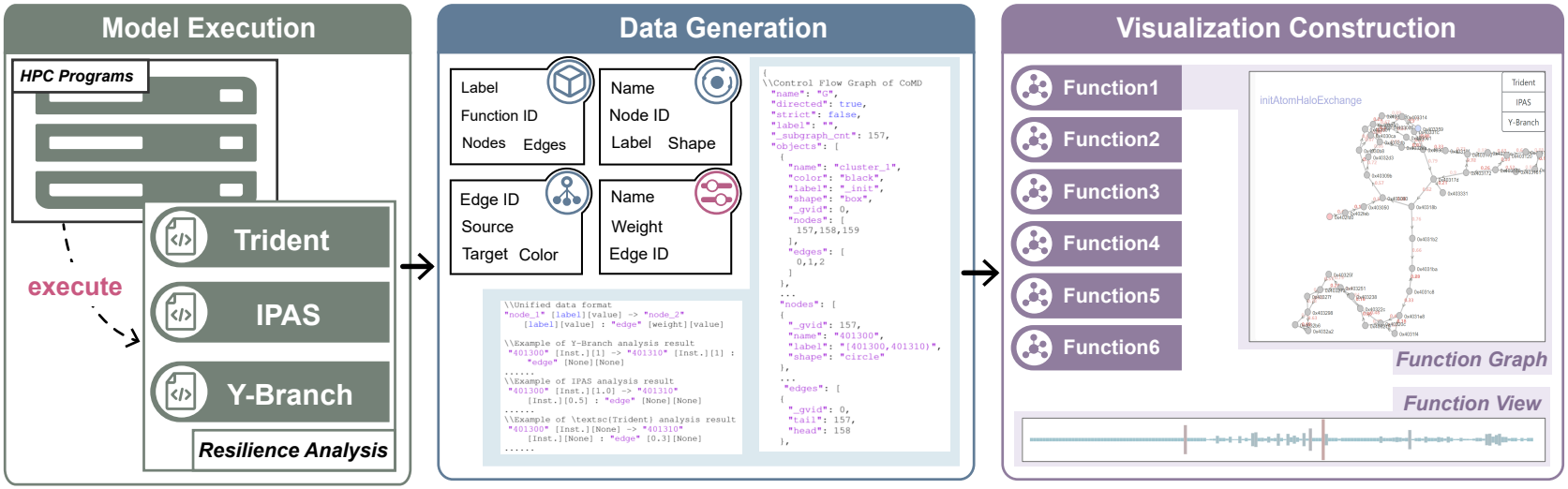

Figure: An overall overview of VISILIENCE

(A) Three resilience analysis models in Resilience Analysis part: TRIDENT [2], IPAS [3] and Y-Branch [4];

(B) Data Generation part generate CFG data , and encodes the resilience analysis results into a unified format;

(C) Visualization Engine takes the formatted data and CFG data as input and outputs an interactive visual interface of the resilience analysis results.

Benefits of using CFG representation

CFG is widely used in compiler optimizations and static analysis tools. VISILIENCE uses CFG as a visualization layout due to the following reasons:

- A CFG not only comprises functions, basic blocks, instructions, and execution paths of the program but also shows the dependency between all these elements. Accommodation of multiple-level information and dependency can help programmers understand resilience better.
- The data size of an HPC program CFG is hugely smaller than dynamic program states, which reduces the resilience analysis time and overhead.
- CFG is a graphical representation of a program. It naturally visualizes how execution traverses a program intuitively.
- Weights are added to the edges in the CFG so that one more data dimension could be represented.

Data Generation

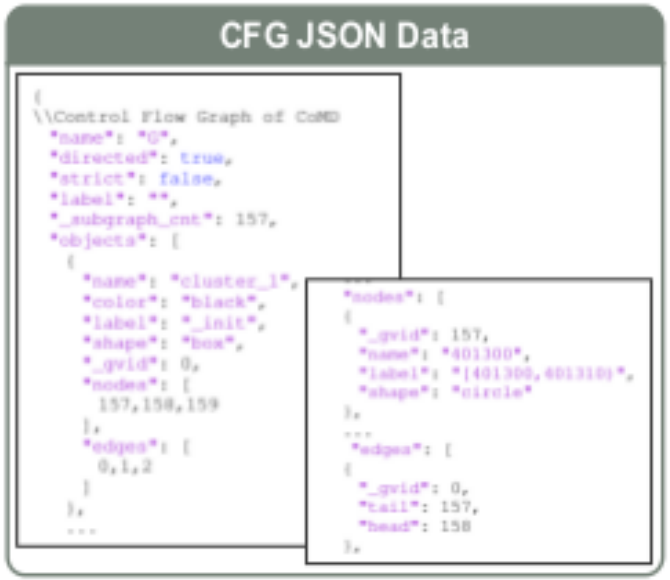

Figure: CFG json data

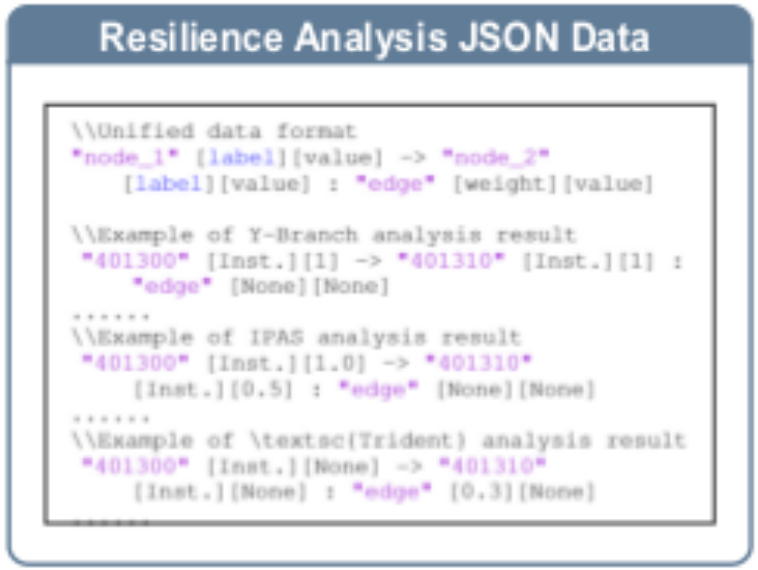

Figure: Resilience analysis json data

Visual Encoding

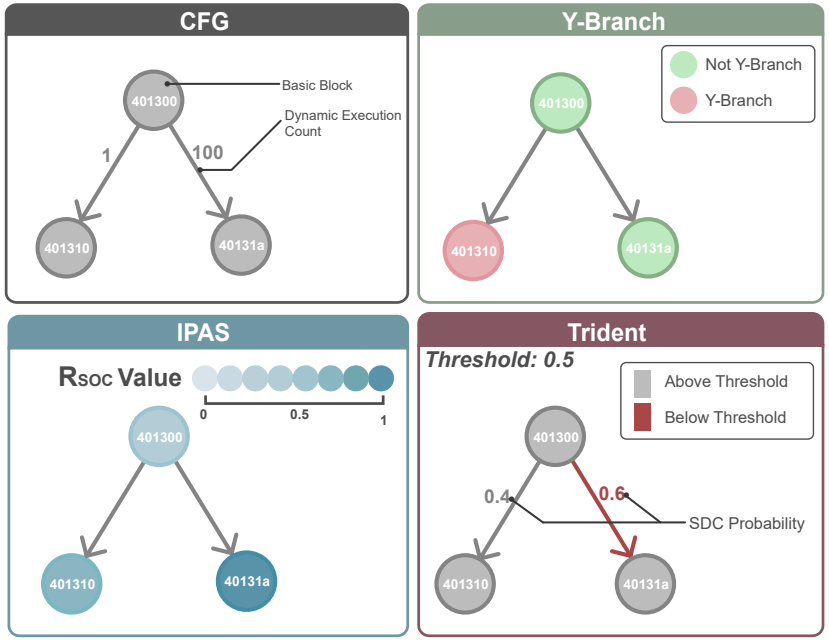

Figure: Visual encoding diagrams

- CFG
  - Nodes: Basic Blocks
  - Edges: Control Flows
- Y-Branch
  - Nodes: Green/Red -> Y-Branch/not
  - Edges: Control Flow
- IPAS
  - Nodes: Darkness -> Rates of SOC inst.
  - Edges: Control Flow
- Trident
  - Nodes: Basic Blocks
  - Edges: Control Flow
  - Weights: SDC probability

PRDC 2023

Visualization Engine Design

DC1. Imitate the basic block interface to generate the layout.

Although the sequence of basic blocks is available, a layout that resembles the basic block interface is still uncertain.

DC2. Enhance the scalability to support real-world data.

The layout generation process module handles both the generation and rendering tasks. In contrast, many entities will be parsed prior to the graph rendering. It is a tough job to parse and generate the position of each node simultaneously. Specifically, powerful hardware is needed when the velocity force *vx* and *vy* of each node are hard to iterate if the number of entities gets 5000 or more.

DC3. Visualize the connection status between different clusters.

The anomalous edges exist both within the cluster and the connection between adjacent clusters. The authoring system must handle the diff array between two different clusters prior to the generation of the graph layout and the re-rendering stage.

Visualization Engine System Workflow

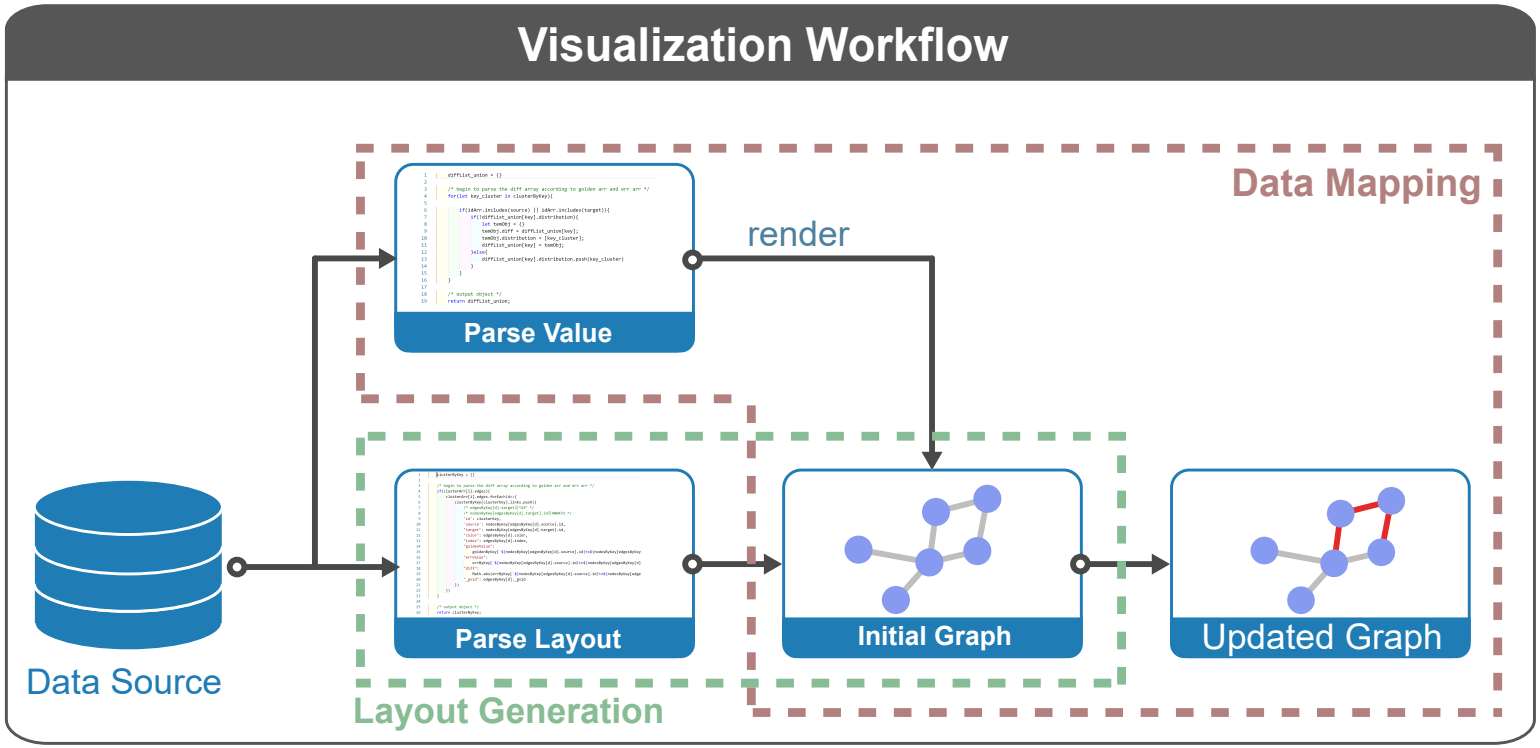

Figure: The workflow of our visualization system incorporates two stages. The layout file will be handled and generated the information of nodes and edges including *vx* and *vy* simulation. Prior to the re-rendering of anomalous mapping.

Interface

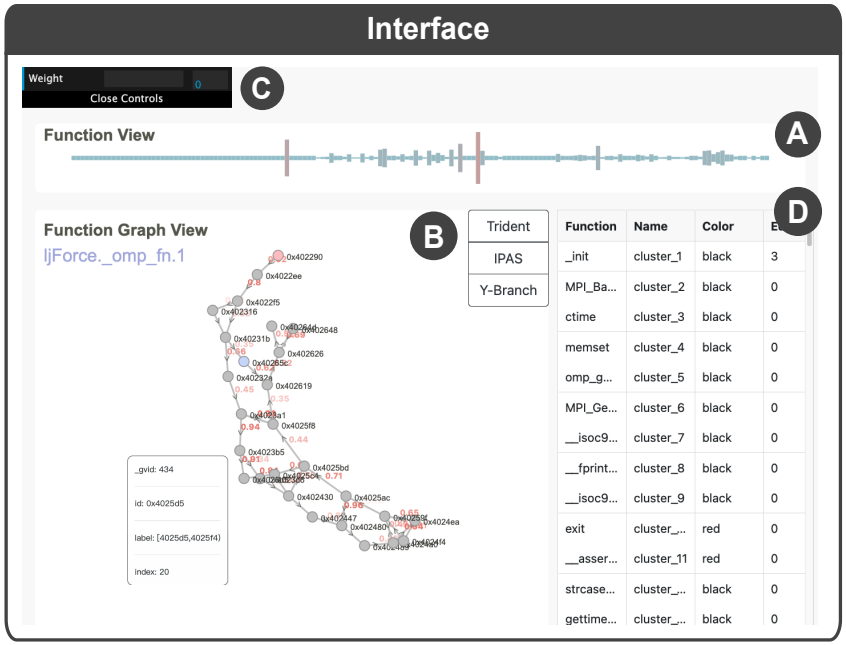

Figure: The interface of VISILIENCE

- A Function view is a series of dots at top represent the functions;
- B The graph is shown in the Graph view and the nodes are basic blocks;
- C Weight threshold is used to set the weight threshold;
- D The functions with specific names are listed in Function List.

Figure: The web link of VISILIENCE

Case Study

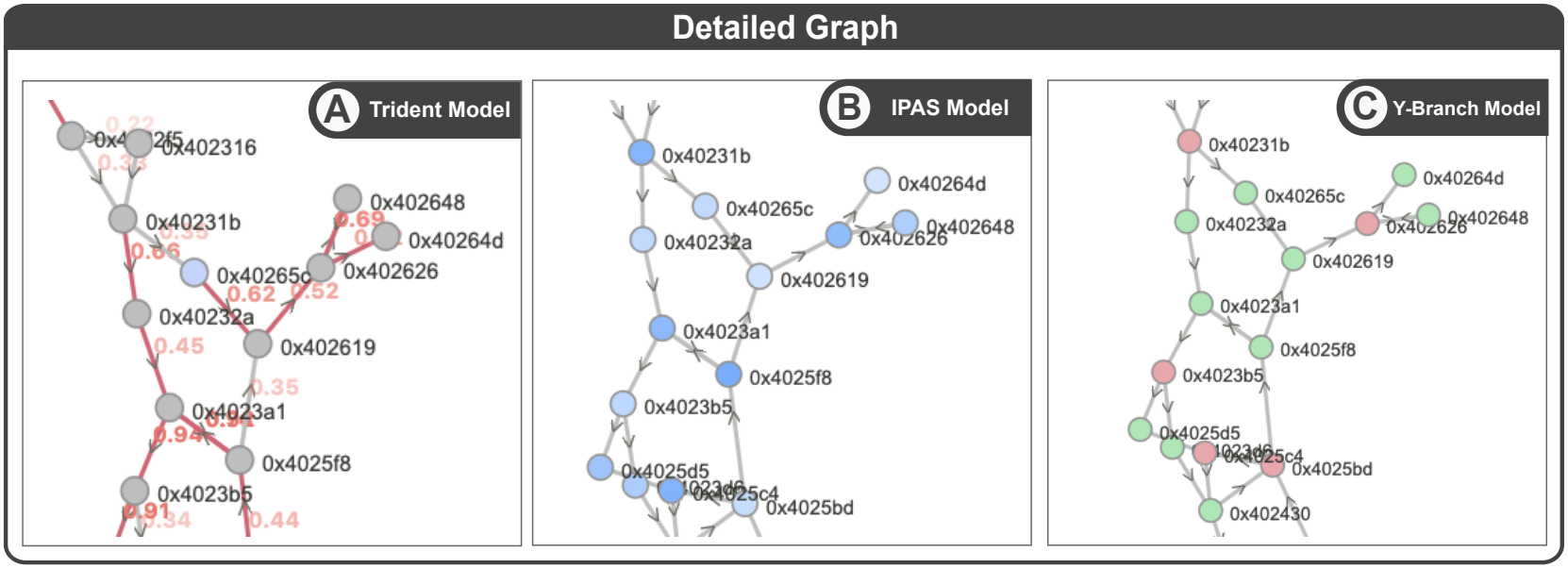

Figure: The snapshots of the partial view of CoMD benchmark under three models

- **Trident:** The weights on the edges are the SDC propagation possibilities between basic blocks.
- **IPAS:** The darkness of the nodes represents the SOC-generating-instruction rate: the darker the colour, the higher the rate.
- **Y-Branch:** Basic blocks in Y-Branch node are green or red, representing Y-branch and non-Y-Branch.

Discussion

Scalability

In the visualization process, the capacity of the virtualization engine in our tool has a limit that each function can accommodate up to 1000 basic blocks.
For our experiments and benchmarks, the number of basic blocks is far less than 1,000.

Applicability

Besides, Visilience can be combined with other performance profile tools, such as HPCtoolkits [5], HPCtraceViewer and so on.

We propose a novel interactive visualization framework, VISILIENCE

It supports three resilience analysis models and enables the interpretable resilience analysis results between different analysis models through several human-computer interactions that help users understand the resilience data.

The VISILIENCE Visualization Engine is based on the Control Flow Graph (CFG)

can accommodate information from the instruction level to the function level. VISILIENCE can be combined with other static analysis tools which use CFGs. The resilience analysis outcomes based on CFG can directly guide the compiler optimizations.

A unified JSON data format to Visualize Engine in VISILIENCE

The unified data interface can bridge the gap of understanding the differences between resilience analysis models.

PRDC 2023

THANK YOU FOR YOUR ATTENTION!

You are welcome to contact:

Hailong Jiang: hjiang13@kent.edu, Kent State University
Shaolun Ruan: haywardryan@foxmail.com, Singapore management university
Bo Fang: bo.fang@pnnl.gov, Pacific Northwest National Laboratory
Yong Wang: yongwang@smu.edu.sg, Singapore management university
Qiang Guan: qguan@kent.edu, Kent State University

PRDC 2023

References I

[1] Jiesheng Wei et al. “Quantifying the accuracy of high-level fault injection techniques for hardware faults”. In: **2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks**. IEEE. 2014, pp. 375–382.

[2] G. Li et al. “Modeling Soft-Error Propagation in Programs”. In: **2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)**. June 2018, pp. 27–38. DOI: 10.1109/DSN.2018.00016.

[3] Ignacio Laguna et al. “IPAS: Intelligent Protection Against Silent Output Corruption in Scientific Applications”. In: CGO 2016. 2016.

[4] Nicholas Wang, Michael Fertig, and Sanjay Patel. “Y-branches: when you come to a fork in the road, take it”. In: **Parallel Architectures and Compilation Techniques, 2003. PACT 2003. Proceedings. 12th International Conference on** (2003).

References II

[5] Xu Liu and John Mellor-Crummey. “A Data-Centric Profiler for Parallel Programs”. In: **Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis**. SC ’13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: 10.1145/2503210.2503297. URL: https://doi.org/10.1145/2503210.2503297.