Contention-aware Performance Modeling for Diversely Heterogeneous Edge and Cloud Systems

Ismet Dagli
Colorado School of Mines
Computer Science Department
Golden, CO, USA
ismetdagli@mines.edu

Andrew Depke
Colorado School of Mines
Computer Science Department
Golden, CO, USA
andrewdepke@mines.edu

Andrew Mueller
Colorado School of Mines
Computer Science Department
Golden, CO, USA
mueller@mines.edu

Sahil Hassan
University of Arizona
Electrical and Computer Engineering
Tucson, AZ, USA
sahilhassan@arizona.edu

Ali Akoglu
University of Arizona
Electrical and Computer Engineering
Tucson, AZ, USA
akoglu@arizona.edu

Mehmet E. Belviranli
Colorado School of Mines
Computer Science Department
Golden, CO, USA
belviranli@mines.edu

ABSTRACT

Diversely Heterogeneous System-on-Chips (DH-SoC) are increasingly popular computing platforms in many fields, such as autonomous driving and AR/VR applications, due to their ability to effectively balance performance and energy efficiency. Having multiple target accelerators for multiple concurrent workloads requires a careful runtime analysis of scheduling. In this study, we examine a scenario that mandates several concerns to be carefully addressed: 1) exploring the mapping of various workloads to heterogeneous accelerators to optimize the system for better performance, 2) analyzing data from the physical world in runtime to minimize the response time of the system 3) accurately estimating the resource contention by workloads during runtime since there will be concurrent operations running under the same die, and 4) deferring the operation to the cloud for computationally more demanding operations such as continuous learning or real-time rendering, depending on the complexity of the computation. We demonstrate our analysis and approach on a VR project as a case study by using NVIDIA Xavier NX Edge DH-SoC and a server equipped with NVIDIA GeForce RTX 3080 GPU and AMD EPYC 7402 CPU.

KEYWORDS

Performance modeling; Edge-Cloud systems; Heterogeneous accelerators

ACM Reference Format:
Ismet Dagli, Andrew Depke, Andrew Mueller, Sahil Hassan, Ali Akoglu, and Mehmet E. Belviranli. 2023. Contention-aware Performance Modeling for Diversely Heterogeneous Edge and Cloud Systems. In *Proceedings of the 3rd Workshop on Flexible Resource and Application Management on the Edge (FRAME '23), June 20, 2023, Orlando, FL, USA*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3589010.3594889

FRAME '23, June 20, 2023, Orlando, FL, USA.

ACM ISBN 979-8-4007-0164-1/23/06…$15.00
https://doi.org/10.1145/3589010.3594889

1 INTRODUCTION

Predicting the performance of applications accurately on different computer architectures has become increasingly complex [3, 5, 8, 18] due to the wide variation in design parameters for intra-device (*i.e.*, type of processing units (PUs), number of cores in a PU, and memory) and inter-device (*i.e.*, network bandwidth and edge-cloud systems). These predictions play a critical role in decision-making for task management, resource allocation, runtime choices, and system design. For accurate performance prediction, models and tools should be precise, flexible, and efficient while being easy to scale, develop, and maintain across various domains. Otherwise, incorrect predictions could lead to wasted resources, such as time and energy.

To address the demanding performance and energy needs of emerging application domains such as computer vision, AR/VR, and autonomous systems, DH-SoCs combine general-purpose and specialized processors such as machine learning accelerators and audio/video codecs to achieve higher efficiency. Compared to general-purpose processors, DH-SoCs provide significantly better performance and energy efficiency for targeted areas of use. For example, autonomous drones perform several computationally intensive tasks, and DH-SoCs employ a diverse range of accelerators that can efficiently execute those tasks: vision accelerators for perception workloads, GPUs and/or FPGAs for localization and motion planning, AI accelerators for deep learning operations, and CPU cores for communication and scheduling [13, 22].

Efficiently mapping different workloads onto heterogeneous computing units to achieve optimal performance or energy efficiency is a challenging task. Effectively utilizing computing resources and optimally managing them is key to achieving efficient use of DH-SoCs. This involves considering several factors including: 1) *Collaborative Edge and Cloud Systems:* Offloading some workloads to cloud servers may generate significant speed-ups when DH-SoCs lack high-performance processing units. However, accurately estimating the underlying performance model is necessary to efficiently utilize the cloud system to ensure computation time on the cloud is less than on the DH-SoC. 2) *Computational characteristics:* Heterogeneous accelerators exhibit varying execution time characteristics for different workloads. An optimal schedule

must target assigning workloads to the best-performing accelerator while also considering other candidate accelerators. 3) *Resource contention:* Shared resources, such as memory, will be simultaneously accessed by multiple PUs on the system, resulting in significant slowdowns that should be minimized. 4) *Data-dependent control flow graph (CFG):* The CFG can vary based on real-world scenarios, such as a moving object that requires fast object detection in an autonomous system or a sudden movement of a VR glass that necessitates complex but low-latency rendering. Minimizing the system's response time is crucial to ensure system safety or improve quality of service (QoS).

There have been a wide range of efforts to model heterogeneous execution. However, research focused on heterogeneous scheduling for systems at the higher end of the diversity spectrum has either made certain assumptions to mitigate the complex and extensive design space that must be considered or failed to present a comprehensive model applicable to various scenarios. We can categorize them into three major classes: black-box, white-box, and hybrid. The first approach [7, 15, 17] mostly relies on heuristics to reduce the complexity of expensive experiment costs but still suffers from high sampling costs. On the other hand, white-box approaches [16, 21, 23] model numerical options but require expensive computation for profiling and significant effort for the implementation to explore the interactions among design candidates. Hybrid methods, combining white- and black-box approaches, [1, 4, 6] are explored for online modeling and the goal is to eliminate the dependency of performance modeling tools for a single feature, such as entirely static dataflow analysis or perfect loop modeling, by analytically and/or empirically combining multiple features onto a model. However, none of the approaches listed are generalizable and flexible enough to represent and model the computational complexity presented by deploying DH-SoCs on an edge-cloud systems. Therefore, there is a pressing need for advanced models and tools that can accurately predict the performance of workloads at runtime across different computer architectures and can be easily adapted to various domains.

Our preliminary research has shown that different stages of a workload have unique computation characteristics depending on the platform and accelerator being used. So, a holistic mapping approach is required in order to exploit the advantages of heterogeneity and fully utilize the system's resources. To address this, we propose a graph-based hardware representation scheme, which is built upon the FLAME model [2], that enables the creation of flexible and scalable abstractions of DH-SoC based edge-cloud systems. In this model, wires (i.e., interconnects) are represented by edges, and any component connected to an interconnect, such as CPU cores, memory controllers, and arithmetic logic units (ALU), is represented by a node. We further analyze the characteristics of different workloads under different input rates when run on heterogeneous PUs and profile the workload's behavior. Our work identifies core factors to develop a performance model that captures complex application and hardware behavior at runtime. We demonstrate our analysis and approach on a VR application case study using the NVIDIA Xavier NX Edge DH-SoC and a server equipped with NVIDIA GeForce RTX 3080 GPU and AMD EPYC 7402 CPU.

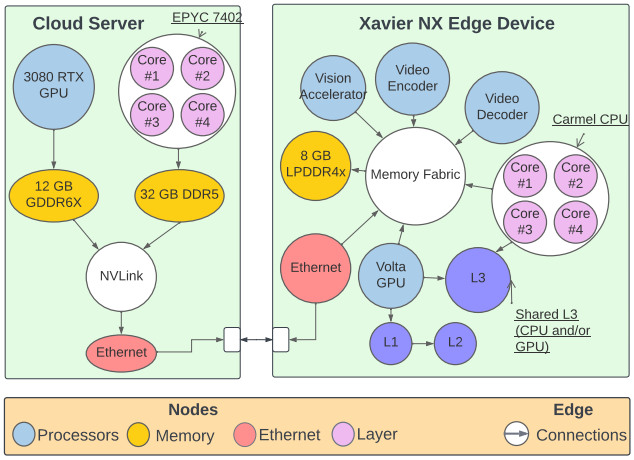

Figure 1: Graph-based hardware representation of our targeted edge-cloud platform.

2 METHODOLOGY

In this section, we design novel graph-based hardware and application representations and a modular performance modeling interface. We generate schedules by using CEDR [11] at runtime.

2.1 Graph-based Hardware Representation

Graphs have been extensively adopted in electronic design automation (EDA) as a core tool to represent the underlying circuit hardware [9, 10]. The most common practice is to represent logic gates with nodes and connecting wires as edges. The resulting graph often contains millions of nodes and edges, and efficient use of heuristical graph algorithms during different phases of EDA has been a heavily focused topic over decades of industrial and academic research. On the other hand, using graph-based representations in performance modeling to represent higher-level PUs that operations can be mapped onto, i.e., scheduling, has not been well studied. Representing heterogeneous systems accurately is a challenging task due to their increasing complexity.

To address this issue, we propose a novel, multi-level-graph-based scheme to represent the HW components in DH-SoCs. The novelty of this approach lies in applying a graph model of hardware to analytically predict its performance. This approach is also supported by a previous work, FLAME [2], using a connected multi-layer graph topology to describe the hierarchical interactions between multiple levels of HW abstraction.

In Figure 1, we illustrate an example graph-based representation of an off-the-shelf edge-cloud platform. Nodes are used to represent different elements of the system, which are a CPU cores, memory elements, bus controller, network port to the server, domain-specific accelerators, or any other element of a connected computing system. Every node is accompanied by information about the computing element, known as *traits*. The traits required for each hardware model depend on the prediction method the user intends to use. Edges, in turn, represent the physical connections in the system that shows where information can flow. This high-level abstraction of

the hardware design allows us to utilize multiple performance models, such as Roofline, machine learning approaches [14], analytical approach [19].

In order to make the model more flexible with a controllable degree of representation granularity, a HW component can be detailed further to whatever level of specification the user desires to specify. As illustrated in Figure 1, the graph-based model is flexible enough to represent the detail levels including multiple levels of cache and down to the core-level representation of a Carmel CPU. This level of generic abstractions of hardware components allows the user to specify the hardware they wish to predict to the level of details that they have access to. While, in general, specifying with as much granularity as possible will yield a better prediction accuracy, this may not be possible for every component in a DH-SoC, such as black-box DSAs. Thanks to multi-layer representation capability, we can model a system composed of components with varying degree of HW details available to the public.

2.2 Generalized Application Representation

Our model assumes that applications are broken down into smaller tasks. Tasks are made up of two major parts, a list of attributes and a list of dependencies. Figure 2 shows an example of an application model, stored as a JSON File. Each task is an example of our case study cloud rendering VR project, which will be elaborated on in Section 3. The attributes are summarized as a list of PUs the task can run and the execution time when the task runs on that particular PU (indexes are aligned with PU entry), and the dependencies among tasks. Our scheduler opts for which task will be executed on which device, as explained in Section 2.3. Dependencies can be arranged into a CFG and all CFGs should be directed acyclic graphs (DAG). A CFG contains one source and one sink to indicate a clear process start and exit. The direction of a CFG is used to show the flow of tasks, while the acyclic nature is necessary to prevent cyclic dependencies.

```
{"Xavier NX & Edge profiling":{
      "Start":{
        "depends":[]
      },
      "Motion Prediction":{
        "PU" :[CPU, GPU],
        "time":["279ms", "75ms"],
        "depends":["start"]
      },
      "Client Networking":{
        "PU" :[CPU],
        "time":["0.253ms"],
        "depends" : ["Motion Prediction"]
      }
      "Server Rendering+Encode":{
        "PU" :[Cloud CPU],
        "time":["26.2ms"],
        "depends" : ["Client Networking"]
      },
      "Client Decode":{
        "PU" :[CPU, GPU],
        "time" : ["3.26", "0.5ms"],
        "depends" : ["Server Rendering"]
      },
      "Client Reproject":{
        "PU" :[CPU(cv), CPU(vpi), GPU(vpi), VIC ],
        "time" : ["46.47ms", "61.37ms", "65.63ms", "79.68ms"],
        "depends" : ["Server Rendering"]
      },
}
```

Figure 2: An example application model for cloud-based VR

Many other attributes (such as arithmetic intensity, memory access amounts, memory throughput, input type and size, etc.) are omitted for the sake of simplicity. Arithmetic intensity is useful for roofline prediction whereas we also utilize memory throughput to detect memory contention by adapting the model proposed by PCCS [25]. This model represents the slowdown encountered by each PU based on the external memory pressure created by other processors.

Cost Function (Traverser): The traverser is the top-level function bringing the hardware and application model together to drive the predictions. The traverser is developed to determine and manage the set of tasks available to run and execute them on the machine model. A task is determined as available to run when all of its dependencies are met. From the list of available tasks, the traverser assigns as many of these tasks as possible to match with compatible PUs. The traverser stops assigning tasks when it runs out of available PUs or tasks available to assign. The traverser then calls the predict function with the set of assigned tasks and the hardware model. The predict function is designed to support multiple kinds of prediction including the Roofline prediction model [24], hardware profiling, and PCCS contention modeling [25]. The prediction will return how long it would take for each task to finish executing on the given resources (not only PUs but also other hardware components affecting the execution time) it is assigned to. The quickest task to execute is removed from the available tasks and all tasks that are running are updated with a completion percentage. Then, the traverser repeats the cycle of finding new available tasks, assigning, and predicting until all tasks are completed. The traverser replaces the cost functions produced by analytical models. Our traverser is called every time the scheduling algorithm needs to calculate the completion time of a subset of the CFG. Overall, the traverser is responsible to account for varying amounts of slowdowns during the lifetime of tasks at runtime.

2.3 Runtime Environment

To realistically study the impact of different task scheduling decisions on a heterogeneous pool of resources, there is a need for a heterogeneous runtime framework that can be deployed on off-the-shelf SoCs. This runtime should enable users to study the impact of scheduling policies on execution latency, as well as resource and memory contention. In addition, this framework should be flexible and adaptable to integrate novel scheduling heuristics based on the predictions derived from the traverser, such that scheduling can be performed at runtime. We leverage the Compiler-integrated Extensible DH-SoC Runtime (CEDR [1]) for this purpose. CEDR [11] is a Linux-based runtime management environment, enabling seamless execution of real-world applications on heterogeneous resource pools over commercial off-the-shelf SoC platforms such as Xilinx ZCU102 MPSoC, NVIDIA Jetson AGX, and Odroid XU4. CEDR being implemented in the Linux user-space provides portability across a wide range of platforms, minimizing migration efforts for the involved developers. With this runtime in place, we perform rapid scheduling heuristics and diverse workload compositions. We can craft execution scenarios such that certain applications are assigned

[1]Available at: https://github.com/UA-RCL/CEDR

to run on a specific subset of available PUs of the platform. On the other hand, we can also emulate execution scenarios with maximum available PUs and allow all applications to run unconstrained on any of these resources. Furthermore, the existing schedulers in CEDR allow us to study the impact of different scheduling decisions at the CFG node level. Under different application execution scenarios, we can study the impact/overhead of migrating a node of an application to a different resource (local, or remote server), using the available hardware (PAPI) counters. These counters provide a profile of execution latency and memory access overhead in terms of instruction and cycle counts, cache loads, and misses, enabling in-depth analysis of application node execution.

CEDR has also been deployed in an environment with edge-based frontend and cloud-based backend compute capability for data science workflows [20]. In order to meet the individual performance requirements of multiple dynamically arriving data science pipelines, the resource manager needs to have a global view of the system resources along with data transfer cost between system resources to be able to determine the modality of the execution and decide whether a specific task in a given workflow should be executed on the resource-limited edge node or resource-rich cloud compute node. CEDR, when deployed as a middleware between the edge and cloud-based compute nodes, provides the schedulers with expected data communication between front and backend by processing the overhead of a node on heterogeneous PUs along with resource status to identify congestion. This allows seamlessly launching a given task in a workflow on local resources or initiating the process of sending the task and its data to the remote platform for processing. With this dynamic front and backend scheduling and processing capability, CEDR offers to realize a multi-level cloud-edge execution platform that is able to utilize system resources effectively while meeting the performance constraints of scientific workflows.

3 CLOUD-BASED VR RENDERING

3.1 Motivation

Virtual reality (VR) rendering has been a hot topic in immersive entertainment, architecture visualization, etc. One of the most important bottlenecks in VR applications is the massive hardware and real-time performance demands while also maintaining a high-fidelity experience. To achieve this, we design a Cloud Rendering VR project to apply our hardware and application representation and evaluate our methodology in this scenario. In this scenario, we offload heavy-duty frame rendering jobs to a cloud-based server and ship the rendered frames back to an edge device in real time. Using this technique, a very high-fidelity experience can be achieved while keeping hardware costs down, however the primary issue with this approach is the latency induced. In order to offset the latency, we employ speculative rendering, where we pre-render the upcoming frames by predicting upcoming user inputs. The inputs are assumed to be provided via human body pose, which is a commonly used technique in VR-based applications.

3.2 Implementation

For the implementation, we use NVIDIA Xavier NX Edge DH-SoC as an edge device and a server system equipped with NVIDIA GeForce

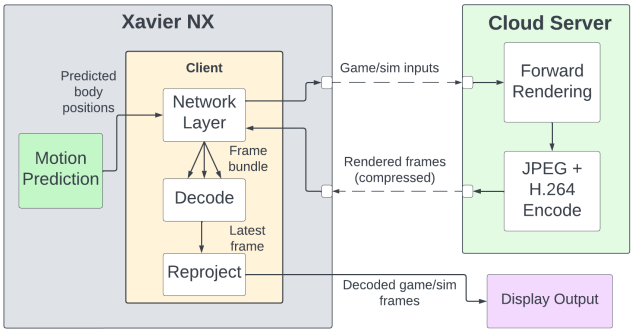

Figure 3: High-level architecture of cloud-based VR rendering

RTX 3080 GPU and AMD EPYC 7402 CPU. The CFG and the tasks assigned to the edge device and the server are illustrated in Figure 3. For our architecture, we start the execution by using a long short term memory (LSTM)-based machine learning model [12] for generating human full body pose predictions in the near time periods, learning from recent sensor data and prior bodily positions. This model, called the pose predictor, outputs statistical distributions for bone orientations in BODY25 format, then computes the final relative space rotation and orientation of the human head. This predicted orientation of the human head is then forwarded to our next component, the client. The client handles all interfacing with the server, so it begins by sending the latest motion information from the client over to the server for rendering. In parallel, it continuously downloads a compressed stream of fully rendered frames produced on the server, where it will then decode and perform a spatial reprojection to compensate for prediction discrepancies. This spatial reprojection step is comprised of running the perspective warp algorithm on the frame, followed by a cropping operation, in order to hide mispredictions. Since this stage happens on the client, we sample the latest headset pose data using local sensors, and use this information for our spatial warp. Lastly, the reprojected frame can then be displayed to the user. In the server component, we capture in motion information from the client, update the game or simulation state accordingly, and then render a new frame and send it over the network to the client after performing H.264 compression.

3.3 Computational Heterogeneity

In order to deploy our real-time use case on the diversely heterogeneous computing hardware, we need to adaptively change the execution schedule of our solution in real-time. In Figure 2, we demonstrate an example scheduling configuration for this project that displays all of the various execution platform options we have for each individual component involved. Looking at the client decoding piece, we can see that we have opportunities to run our H.264 frame decoding stage on either the CPU or by using a hardware decoder (NvDec for Xavier NX). We separately mark the available common PUs (such as CPU or GPU) as the cloud's PU. The choice between using each largely depends on computing resources available to the given platform, earliest finish time, shared or dedicated memory, etc. On the NVIDIA Xavier NX device, there is a shared memory fabric interconnecting all of the system's accelerators, so choosing to send the frame data to the NvDec accelerator and back

has contention slowdown that can change how we schedule the component and others on the system.

3.4 Results

The Cloud Rendering VR project highlights a real-world use case and applicable to our heterogeneous modeling solution. In our testing, we observed that there is a non-linear scaling relationship between the different accelerators available on the Xavier NX DH-SoC, as seen in Figure 4. From this data, we can further assist our scheduling decisions to optimize for a broader range of hardware. We observe that using dynamic and adaptive scheduling for our various compute tasks shows promise as a means of deploying to wide range of real hardware. We believe that continuing with this work and creating more scheduling options for tasks such as our motion prediction or frame decoding stages will enable more effective means of execution on all platforms.

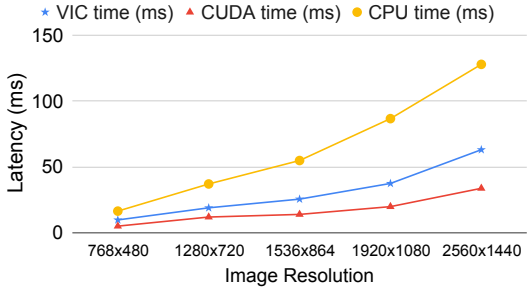

Figure 4: Effect of rendering resolution sizes on reprojection latency, measured on the Xavier NX platform

4 CONCLUSION

Diversely heterogeneous Systems-on-chips (SoCs) are gaining popularity in emerging areas such as autonomous driving and AR/VR applications, primarily due to their high performance and energy efficiency. In this work, we examine mapping concurrently running workloads to a range of accelerators demands a meticulous modeling of performance, including consideration of contention. We present a graph-based hardware model and a query-based application model. We evaluate our methodology on a realistic cloud-based VR rendering scenario.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their feedback. This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF-2124010. Any opinions, findings, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7860. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory (AFRL) and Defence Advanced Research Projects Agency (DARPA) or the U.S. Government.

REFERENCES

[1] Yehia Arafa, Abdel-Hameed Badawy, Ammar ElWazir, Atanu Barai, Ali Eker, Gopinath Chennupati, Nandakishore Santhi, and Stephan Eidenbenz. Hybrid, scalable, trace-driven performance modeling of gpgpus. In *SC*, pages 1–15, 2021.

[2] Mehmet E. Belviranli and Jeffrey S. Vetter. Flame: Graph-based hardware representations for rapid and precise performance modeling. In *IEEE Design, Automation Test in Europe Conference Exhibition (DATE)*, 2019.

[3] Arnamoy Bhattacharyya and Torsten Hoefler. Pemogen: Automatic adaptive performance modeling during program runtime. In *PACT*, 2014.

[4] Arnamoy Bhattacharyya, Grzegorz Kwasniewski, and Torsten Hoefler. Using compiler techniques to improve automatic performance modeling. In *PACT*, pages 468–479. IEEE, 2015.

[5] Marcin Copik, Alexandru Calotoiu, Tobias Grosser, Nicolas Wicki, Felix Wolf, and Torsten Hoefler. Extracting clean performance models from tainted programs. In *PPoPP*, pages 403–417, 2021.

[6] Yuyang Jin, Haojie Wang, Runxin Zhong, Chen Zhang, and Jidong Zhai. Perflow: A domain specific framework for automatic performance analysis of parallel applications. In *PPoPP*, pages 177–191, 2022.

[7] Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, Alexander Grebhahn, and Sven Apel. Tradeoffs in modeling performance of highly configurable software systems. *Software & Systems Modeling*, 2019.

[8] Seyong Lee, Jeremy S Meredith, and Jeffrey S Vetter. Compass: A framework for automated performance modeling and prediction. In *ICS*, pages 405–414, 2015.

[9] Daniela Sánchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, and Wolfgang Ecker. A survey of graph neural networks for electronic design automation. In *2021 ACM/IEEE 3rd Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6. IEEE, 2021.

[10] Yuzhe Ma, Zhuolun He, Wei Li, Lu Zhang, and Bei Yu. Understanding graphs in eda: From shallow to deep learning. In *Proceedings of the 2020 International Symposium on Physical Design*, pages 119–126, 2020.

[11] Joshua Mack, Sahil Hassan, Nirmal Kumbhare, Miguel Castro Gonzalez, and Ali Akoglu. CEDR: A Compiler-Integrated, Extensible DSSoC Runtime. 22(2), 3 2023.

[12] Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2891–2900, 2017.

[13] Junayed Pasha, Zeinab Elmi, Sumit Purkayastha, Amir M Fathollahi-Fard, Ying-En Ge, Yui-Yip Lau, and Maxim A Dulebenets. The drone scheduling problem: A systematic state-of-the-art review. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[14] Thanh-Phuong Pham, Juan J Durillo, and Thomas Fahringer. Predicting workflow task execution time in the cloud using a two-stage machine learning approach. *IEEE Transactions on Cloud Computing*, 8(1):256–268, 2017.

[15] Marcus Ritter, Alexandru Calotoiu, Sebastian Rinke, Thorsten Reimann, Torsten Hoefler, and Felix Wolf. Learning cost-effective sampling strategies for empirical performance modeling. In *IPDPS*. IEEE, 2020.

[16] Larissa Schmid, Marcin Copik, Alexandru Calotoiu, Dominik Werle, Andreas Reiter, Michael Selzer, Anne Koziolek, and Torsten Hoefler. Performance-detective: automatic deduction of cheap and accurate performance models. In *ICS*, 2022.

[17] Norbert Siegmund, Alexander Grebhahn, Sven Apel, and Christian Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[18] Jingwei Sun, Guangzhong Sun, Shiyan Zhan, Jiepeng Zhang, and Yong Chen. Automated performance modeling of hpc applications using machine learning. *IEEE Transactions on Computers*, 69(5):749–763, 2020.

[19] Nathan R Tallent and Adolfy Hoisie. Palm: Easing the burden of analytical performance modeling. In *ICS*, pages 221–230, 2014.

[20] Genoveva Vargas-Solar, Ali Akoglu, and Md Sahil Hassan. JITA4DS: Disaggregated execution of Data Science Pipelines between the Edge and the Data Centre. *Journal of Web Engineering*, 21(01):1–26, 2021.

[21] Miguel Velez, Pooyan Jamshidi, Norbert Siegmund, Sven Apel, and Christian Kästner. White-box analysis over machine learning: Modeling performance of configurable systems. In *ICSE*, pages 1072–1084. IEEE, 2021.

[22] Yifan Wang, Shaoshan Liu, Xiaopei Wu, and Weisong Shi. Cavbench: A benchmark suite for connected and autonomous vehicles. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 30–42. IEEE, 2018.

[23] Max Weber, Sven Apel, and Norbert Siegmund. White-box performance-influence models: A profiling and learning approach. In *ICSE*. IEEE, 2021.

[24] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.

[25] Yuanchao Xu, Mehmet Esat Belviranli, Xipeng Shen, and Jeffrey Vetter. Pccs: Processor-centric contention-aware slowdown model for heterogeneous system-on-chips. In *MICRO*, pages 1282–1295, 2021.