STOMP: Agile Evaluation of Scheduling Policies in Heterogeneous Multi-Processors

Augusto Vega[1], John-David Wellman[1], Hubertus Franke[1], Alper Buyuktosunoglu[1], Pradip Bose[1]
Aporva Amarnath[2], Hiwot Kassa[2], Subhankar Pal[2], Ronald Dreslinski[2]
[1]*IBM T. J. Watson Research Center*, [2]*University of Michigan*
Contact email: ajvega@us.ibm.com

***Abstract*—The proliferation of heterogeneous chip multiprocessors in recent years has reached unprecedented levels. Traditional homogeneous platforms have shown fundamental limitations when it comes to enabling high-performance yet-ultra-low-power computing, in particular in application domains with real-time execution deadlines or criticality constraints. By combining the right set of general purpose cores and hardware accelerators together, along with proper chip interconnects and memory technology, heterogeneous chip multiprocessors have become an effective high-performance and low-power computing alternative.**

One of the challenges of heterogeneous architectures relates to efficient scheduling of application tasks (processes, threads) across the variety of options in the chip. As a result, it is key to provide tools to enable early-stage prototyping and evaluation of new scheduling policies for heterogeneous platforms. In this paper, we present STOMP (Scheduling Techniques Optimization in heterogeneous Multi-Processors), a simulator for fast implementation and evaluation of task scheduling policies in multi-core/multi-processor systems with a convenient interface for "plugging" in new scheduling policies in a simple manner. STOMP is part of our DARPA-funded EPOCHS project.

***Keywords*—scheduling algorithms; heterogeneous SoCs**

I. Introduction

Domain-specific *heterogeneous* systems on a chip (SoC) expose a variety of options for task execution, including but not limited to general-purpose cores, graphics processing units (GPUs), and hardware accelerators. Due to this degree of heterogeneity, task scheduling becomes less trivial when compared to homogeneous counterparts. In its simplest form, the scheduler can *statically* map application tasks to fixed execution units in the chip to reduce the complexity associated with run-time scheduling decisions. This approach, however, can limit the scheduler's capabilities in making dynamic decisions that can lead to better performance or efficiency. Therefore, it becomes critical to provide the right set of tools for early-stage prototyping and evaluation of scheduling algorithms ("policies") in heterogeneous systems, enabling enough flexibility and exploration space coverage.

The Domain-Specific SoC (or DSSoC) program [1] under DARPA MTO's Electronics Resurgence Initiative (ERI) is concerned with the problem of designing easily programmable, yet efficient SoCs (for an identified application domain) at low development cost. In particular, effective task scheduling in heterogeneous SoCs is one of the key challenges within this program. The IBM-led project under DSSoC is called: "EPOCHS: Efficient Programmability of Cognitive Heterogeneous Systems." In this context, we created STOMP (Scheduling Techniques Optimization in heterogeneous Multi-Processors), a queue-based discrete-event simulator that enables fast implementation and evaluation of task scheduling policies in heterogeneous systems [23]. It implements a convenient interface to allow users and researchers to "plug in" new scheduling policies in a simple manner and without the need to interact with STOMP's internal code. We conceive STOMP with the following three goals in mind:

- *Flexibility*: it is straightforward to define simulated platforms and applications, and to test new scheduling policies through STOMP's "plug & play" approach.
- *Ease of use*: STOMP's default execution mode allows users to quickly configure and run simulations at the right level of abstraction. It also supports more detailed simulation capabilities for expert users as well.
- *Openness*: the tool is open source and publicly available [23].

STOMP's core queue-based operation approach builds upon the QUTE framework [17]. However, STOMP introduces radically new elements to support the evaluation of heterogeneous SoCs, allowing users to easily configure multi-core/multi-processor systems with varying degrees of heterogeneity. In one of its execution modes, STOMP can be fed with applications represented as directed acyclic graphs (DAGs), which can be either generated in a synthetic manner or from the characterization of real workloads.

STOMP is thoroughly validated against analytical model counterparts (closed-form expressions). For system utilization levels between 10%-90%, the average relative errors for steady-state analysis of waiting times are: 0.50%, 0.83%, and 1.45% (for one, two and three servers, respectively).

The rest of the paper presents STOMP along with evaluation results using autonomous vehicles applications. It also discusses the ongoing implementation of these advanced scheduling policies in real heterogeneous systems.

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. This document is approved for public release: distribution unlimited.

II. STOMP Simulator

Figure 1 presents a simplified view of STOMP with the most relevant components to support *multi-step* task scheduling. Specifically, task scheduling in STOMP involves two steps: application pre-processing and task scheduling, implemented within the `Meta-Sched` and `Task-Sched` modules, respectively.

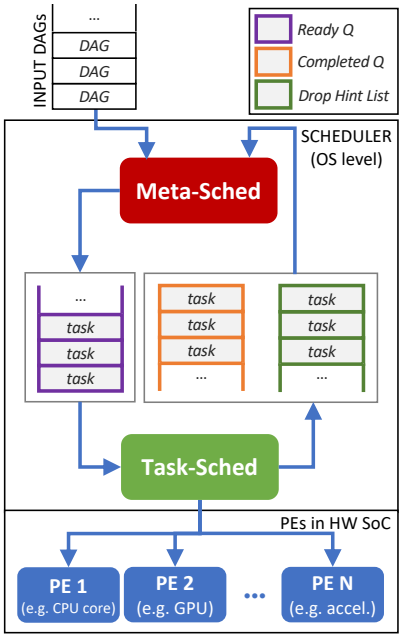

Fig. 1. STOMP overview showing `Meta-Sched` and `Task-Sched`.

`Meta-Sched` continuously analyzes the simulated applications, which are expressed in the form of directed acyclic graphs (DAGs), where nodes represent tasks and edges represent dependencies. A DAG has an associated *execution deadline* and a *criticality level*. `Meta-Sched` uses the DAG's execution deadline to compute tasks' execution deadlines. Similarly, tasks are assigned the criticality level of the DAG they belong to. `Meta-Sched` also computes a *rank* associated to each task, keeps tasks ordered by rank in the *Ready Queue*, and identifies ready (runnable) tasks by tracking dependencies. The *rank* of task i, a concept adopted also in prior works ([22], [26], [27]), can be computed in different ways; e.g. as a function of task i's criticality and deadline:

$$Rank_i = \frac{Criticality_i}{Deadline_i}$$

If task i's criticality is high and/or its deadline is short, then the task will be assigned a relatively large rank and, therefore, will be placed closer to the head of the *Ready Queue* for immediate execution.

`Task-Sched` takes tasks from the *Ready Queue* and assigns them to servers (processing elements) based on the user-specified scheduling policy. `Task-Sched` notifies `Meta-Sched` about completed tasks through the *Completed Queue*. New scheduling policies can be created and "plugged in" as discussed in Section II-B.

STOMP supports two execution modes: *probabilistic* and *realistic*. In probabilistic mode, the arrival rate and service times (execution times) of tasks are determined by configurable probability distributions (e.g. exponentially for the arrival rate). In realistic mode, the tasks and their associated characteristics (like arrival and service times) are loaded from a trace file provided by the user and previously generated, for example, using real profiling data.

Once inserted in the *Ready Queue*, each task has a set of attributes, the following being the most relevant ones:

- *Target servers*: list of servers (processing elements) where the task can execute on and the order of preference. For example, the list $\{accelerator,\ GPU,\ CPU\ core\}$ indicates that the scheduler should first try to place the task in a corresponding accelerator. If an accelerator is not available, then other supported architectures are GPU and CPU core, in that order of preference. Tasks do not necessarily support all the available processing elements — e.g. some tasks may only run on CPU cores, or CPU cores *and* GPUs, etc.
- *Service time*: the list of target servers includes corresponding service (execution) times for each specified processing element. These are *mean* service times used to generate task execution times during simulation. They are ignored in *realistic* execution mode (i.e. when tasks are read from external traces).
- *Power consumption*: similarly, the list of target servers includes corresponding power consumption information for each specified processing element. This information can be used for implementation of power-aware scheduling policies.
- *Execution deadline*: single value associated with the task that indicates the amount of time available for execution, and intended for simulation of real-time constrained applications.

The user can also specify and configure the characteristics of the servers (processing elements). Obvious options include CPU cores, GPUs, and hardware accelerators. At present, STOMP does not support multi-threaded processing elements; in other words, once a task is allocated to a processing element, no other task(s) can be scheduled on it until the currently running task finishes its execution.

A. Configuration Parameters

STOMP simulations are configured through a single JSON file. Some of the most relevant parameters are described below:

- `sched_policy_module`: indicates the scheduling policy to use. For example, `policies.test` uses a policy implemented in the `test.py` file within the `policies` folder. The implementation of new scheduling policies is discussed in Section II-B.
- `max_tasks_simulated`: maximum number of simulated tasks. Only valid with STOMP's *probabilistic* mode.
- `mean_arrival_time`: mean task arrival time. Only valid with STOMP's *probabilistic* mode.
- `arrival_time_scale`: a constant factor to scale `mean_arrival_time`. For example, a 0.5

value will double the task arrival rate (since it halves `mean_arrival_time`); while a 2.0 value will halve the task arrival rate (since it doubles `mean_arrival_time`). Only valid with STOMP's *probabilistic* mode.

- `servers`: definition of servers (processing elements) simulated in the system. For example, the following JSON fragment configures a simulated platform with eight general-purpose cores, two GPUs and one FFT accelerator:

```
"servers": {
        "cpu_core" : {
            "count" : 8
        },
        "gpu" : {
            "count" : 2
        },
        "fft_accel" : {
            "count" : 1
        }
}
```

  A server's name is just an arbitrary string and does not assign any specific characteristics to the server. Instead, execution times and power consumption values are part of each task's information.
- `tasks`: definition of tasks simulated in the system. Only valid with STOMP's *probabilistic* mode. For example, the following JSON fragment creates a simulated FFT task with specific mean service times and associated standard deviations for the simulated heterogeneous platform:

```
"tasks": {
        "fft" : {
            "mean_service_time" : {
                "cpu_core"  : 500,
                "gpu"       : 100,
                "fft_accel" : 10
            },
            "stdev_service_time" : {
                "cpu_core"  : 5.0,
                "gpu"       : 1.0,
                "fft_accel" : 0.1
            }
        }
}
```

  The standard deviation controls the dispersion of the service (execution) time — in other words, it allows the user to set the level of determinism of a task's execution characteristics.
- `input_trace_file`: trace of tasks used for simulation with STOMP's *realistic* mode. The trace also includes the task's arrival time and service times across the different server types in the system.

It is important to mention that the concept of "time" in STOMP is unitless. The user is responsible for providing meaning to the time values used in the configuration file — e.g. a mean service time of "500" units of time could mean 500 μs, or 500 ms, etc.

B. "Plug & Play" Scheduling Policies

The `Task-Sched` module in STOMP (Figure 1) is responsible for assigning ready tasks to servers (processing elements) using a user-specified scheduling policy. The policies can go from very simple decision logic all the way to complex and potentially more "intelligent" ones — eventually using machine learning techniques or other mechanisms for dynamic improvement of the scheduling activities. In STOMP, new policies are constructed by implementing the abstract class `BaseSchedulingPolicy`, shown below:

```
class BaseSchedulingPolicy:

  __metaclass__ = ABCMeta

  @abstractmethod
  def init(self, servers, stomp_stats, stomp_params): pass

  @abstractmethod
  def assign_task_to_server(self, sim_time, tasks): pass

  @abstractmethod
  def remove_task_from_server(self, sim_time, server): pass

  @abstractmethod
  def output_final_stats(self, sim_time): pass
```

Specifically, the task scheduling decision logic is defined within the `assign_task_to_server()` method. The user also has the opportunity to implement initialization and finalization activities as part of the `init()` and `remove_task_from_server()` methods, and to provide policy-specific statistics via `output_final_stats()` to be displayed at the end of the simulation. One possible (illustrative) example of an `assign_task_to_server()` implementation is shown below (in this example, the task is only scheduled to the fastest server type, if available):

```
def assign_task_to_server(self, sim_time, tasks):

  if (len(tasks) == 0):
    # There aren't tasks to serve
    return None

  # Determine task's best scheduling option
  target_server_type =
      tasks[0].mean_service_time_list[0][0]

  # Look for an available server
  for server in self.servers:

    if (server.type == target_server_type
        and not server.busy):

      # Assign task in queue's head to server
```

```
        server.assign_task(sim_time, tasks.pop(0))
        return server

    return None
```

Strictly speaking, scheduling policies are implemented as Python modules, with each new policy in a different Python file. The user indicates the module to load (i.e. the policy to use) through the `sched_policy_module` parameter, as we explain in Section II-A. STOMP's GitHub repository [23] includes examples of scheduling policies that can be used as templates to generate new ones.

III. STOMP Validation

Since STOMP is a queue-based simulator, we address its validation by comparing it against its analytical model counterparts (closed-form expressions). Specifically, we focus on the M/M/k system, as defined by Kendall's notation [25]. An M/M/k system models a single queue with k servers (processing elements), where both arrival and service (computation) times are exponentially distributed. In practice, service times in STOMP are normally distributed; however there are only crude approximations for the M/G/k case which are relatively accurate only for a few constrained cases [9], [13]. This leads us to opt for the M/M/k system to validate the dynamics of the core STOMP simulation engine.

Figure 2 presents the relative error of the average waiting time (steady state analysis) as a function of the system utilization, for the 1-, 2- and 3-server cases (M/M/1, M/M/2 and M/M/3, respectively). The relative error is computed as $|W_{STOMP} - W_{M/M/k}|/W_{M/M/k}$, where W_{STOMP} is the steady-state waiting time generated by STOMP after simulating 1M tasks and $W_{M/M/k}$ is the corresponding waiting time generated using the closed-form formula. In most cases, the relative errors are low. Specifically, for utilization levels between 10%–90%, the average relative errors are 0.50% for M/M/1, 0.83% for M/M/2, and 1.45% for M/M/3. The relative error increases for the 99%-utilization case; it is well known that these formulas are usually not adequate when utilization approaches 100% as the system becomes less stable [9].

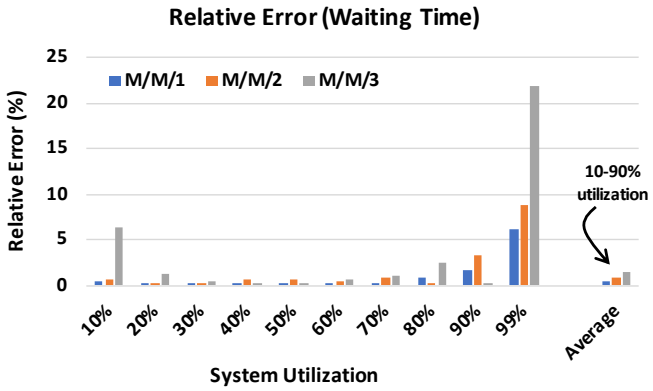

Fig. 2. Relative error of the average (steady state) waiting time as a function of the system utilization.

The accuracy of discrete-event simulators like STOMP is highly dependent on the number of simulated tasks. Usually, when the number of simulated tasks (or "customers") is not "large enough", the system suffers from instability and warming-up conditions that can invalidate the average (steady state) results. Figure 3 presents the relative error of the average waiting time (steady state analysis) as a function of the number of simulated tasks, for the three cases under consideration (M/M/1, M/M/2 and M/M/3). The simulations correspond to 50% system utilization. As we can observe, the relative error decreases when more tasks are simulated. The "right" amount of tasks also depends on the case: 200K tasks is enough to ensure an error smaller than 1% in the M/M/1 case; while at least 400K and 300K tasks are needed in the M/M/2 and M/M/3 cases, respectively, to ensure the same error bound. In the validation campaign conducted in this work (Figure 2), we simulated 1M tasks in all cases to conservatively avoid any possible transient state instabilities during the simulations.

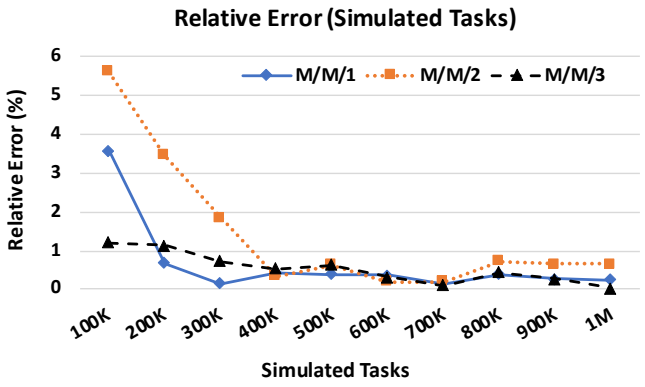

Fig. 3. Relative error of the average (steady state) waiting time as a function of the number of simulated tasks (for the 50% system utilization case).

We are currently working on extending this validation analysis to cases where tasks and servers (processing elements) can be of different types (heterogeneous systems). Due to the lack of closed-form formulas for heterogeneous queueing systems, the validation strategy for these cases will require alternative approaches which may imply the use of third-party (already-validated) discrete-event simulators for the heterogeneous cases.

IV. Evaluation for Real-World Automotive Applications

This section evaluates STOMP using real-world application traces derived from ADSuite [16], an end-to-end autonomous driving (AV) application comprised of kernels like object detection (DET), object tracking (TRA), localization (LOC), mission planning, and motion planning. For DET, we use YOLOv3 [19], a DNN-based detection algorithm, on a series of 7 images derived from the VOC dataset [11]. We use the Tiny-YOLOv3 pre-trained set of weights, which is much faster and lightweight, but less accurate compared to the regular YOLO model. For TRA, we use GOTURN [14], a DNN-based single object tracking algorithm, on a series

of 14 videos in the ALOV++ dataset [21]. For LOC, we use ORB-SLAM [18], a highly-ranked vehicle localization algorithm, on 3 sequences from the KITTI datasets [12]. Further, for our GPU evaluation, we adopt the ORB-SLAM implementation in [3], where the hot paths are rewritten using CUDA. We also obtain timing profile of DET, TRA and LOC on their respective accelerators from [16]. For motion and mission planning, we use the `op_local_planner` and `op_global_planner` [10] kernels in Autoware [15]. The fusion kernel combines the coordinates of the objects being tracked with the AV location. It has a small latency, for which we only consider CPU execution.

To evaluate ADSuite, we first profile (offline) its constituent kernels on an NVIDIA Jetson TX1 board, which is representative of an SoC used in real-world AV systems. This information is then used to simulate a heterogeneous SoC with multiple PEs. We assume that the simulated SoC has variants of the ARM Cortex-A57 CPU and the NVIDIA Maxwell GPUs with 256 CUDA cores, and fixed-function accelerators for certain tasks. We consider a unified memory (shared physical address space) between the PEs in the simulated SoC, since we profiled the applications on the TX1 that has unified memory between the CPUs and GPUs; STOMP, however, is not limited to this specific choice. Table I summarizes the modeled SoC architecture along with other relevant simulation parameters. For the sake of this illustrative evaluation, we prototyped in STOMP a *mission-aware* scheduling algorithm called AVSched that leverages the synergy between the underlying heterogeneous hardware platform and the applications' runtime characteristics to satisfy the growing throughput demand of AVs, while meeting the specified real-time and safety constraints. We choose the following metrics to evaluate AVSched: *mission time* to complete the objective of the mission (e.g. navigation time from location "A" to location "B", while complying with safety requirements of meeting deadline for all critical DAGs); and *fraction (or %) of mission completed* at a given speed before missing the first critical DAG deadline.

TABLE I
SIMULATED CONFIGURATIONS.

Parameter	Values
Simulation Mode	Realistic
Simulated SoC	8 single-core ARM Cortex-A57 CPUs 2 NVIDIA Maxwell GPUs 1 tracking accelerator [16] 1 localization accelerator [16] 1 detection accelerator [16]
Policies	AVSched
Metrics	Mission time; % of mission completed

Figure 4 shows that AVSched achieves up to 8.5×, 5.6×, 5.4× and 5.5× improvement in mission time over CPATH, RHEFT, 2step-EDF and ADS, respectively (state-of-the-art real-time and heterogeneous schedulers). In terms of the % mission completed metric, ADS completes 9%, 11% and 18% of the mission at the maximum safe speed of AVSched for the rural, semi-urban and urban scenarios, respectively, before missing the deadline for the first critical DAG. Furthermore, AVSched is able to achieve 1.7–3.7× better PE utilization over the baseline schedulers.

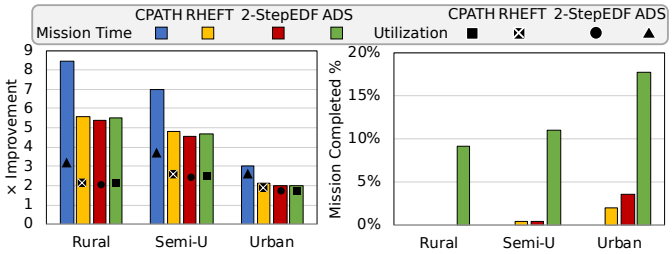

Fig. 4. Comparison of metrics and PE utilization of AVSched against prior-work schedulers for ADSuite. *Left:* Mission speedup and PE utilization improvement of AVSched over the baseline schedulers. *Right:* % Mission completed by baseline schedulers before missing deadline for a critical DAG, while operating at the maximum safe speed achieved by AVSched.

V. Implementation on Real Settings

STOMP allows early-stage prototyping and evaluation of scheduling policies that can be then ported to real heterogeneous chips. As part of this effort, the EPOCHS team is pursuing the Scheduler Library (SL) initiative [24] which provides a user-level application programming interface (API) that allows applications to access the hardware accelerator resources of the system. We use different AV applications as our primary workloads, including our Mini-ERA application [4]. These applications are linked to the SL to provide a scheduler-managed access to the underlying hardware accelerators in our EPOCHS SoC. The SL has three main components (or layers), as indicated in Figure 5:

- **SL API:** abstracts applications from the scheduler's details and provides some degree of portability across applications and execution models.
- **SL Manager:** schedules tasks across heterogeneous PEs to improve metrics of interest (throughput, criticality, efficiency) executing the user-defined scheduling policy. An internal repository stores different binary versions of application kernels targeting different PE types.
- **Hardware APIs:** abstract the scheduler from the heterogeneous hardware through a "plug & play" approach that allows multiple hardware interfaces to coexist — e.g. CPU, GPU, and ESP [2].

The SL is an ongoing project that constitutes a step towards a longer-range vision, that of advanced hardware/software interfaces to enable the revolution promised by hardware specialization.

VI. Related Work

A plethora of work exists on various scheduling policies for heterogeneous architectures. However, most of these scheduling policies are either evaluated on in-house simulators or runtime systems. To the best of our knowledge, STOMP is the first open-source tool for agile evaluation of scheduling algorithms in heterogeneous system that was conceived with "plug & play" flexibility in mind.

Runtime systems like StarPU [6] and Nanos++ [7], [8] provide scheduler frameworks. However it can be extremely tedious to develop and compare scheduling policies in a

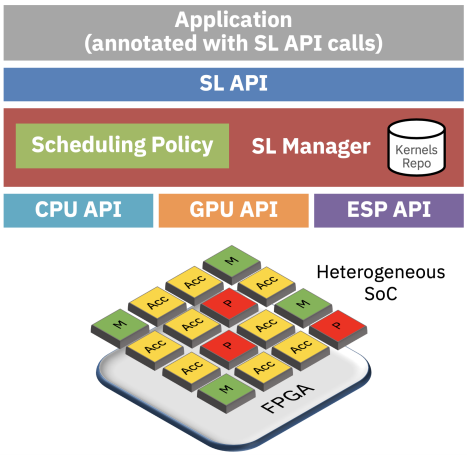

Fig. 5. Scheduler Library (SL) overview.

runtime system that has been developed to perform various kernel operations. Moreover the data structures and models of the runtime system can constrict the generality of a scheduling policy being developed [6].

TaskSim [20] is a simulator for the execution of tasks on decoupled accelerator systems with the capability of scheduling tasks. DS3 is a simulator for heterogeneous SoCs with scheduling and power management features [5]. Both TaskSim and DS3 are full-system simulators and, consequently, agile evaluation and comparison of multiple scheduling algorithms is not necessarily a straightforward process.

VII. Conclusion

This paper presents STOMP, a simulator for fast evaluation of scheduling policies in domain-specific SoCs (developed under the DARPA-funded DSSoC program). Using real-world AV applications, we show results of a proposed "smart" scheduling policy (AVSched) that improves critical metrics, like mission time for AVs. We also discuss the ongoing implementation of advanced scheduling policies in real heterogeneous systems through the Scheduler Library (SL) initiative within our EPOCHS project.

Acknowledgments

The authors gratefully acknowledge the technical contributions of many students, postdoctoral researchers and IBM research staff members that have collectively worked to make this an exciting and successful research endeavor.

References

[1] "Domain-specific system on chip (DSSoC)," https://www.darpa.mil/program/domain-specific-system-on-chip.
[2] "ESP: the open-source SoC platform." [Online]. Available: https://www.esp.cs.columbia.edu
[3] "ORB-SLAM2-GPU." [Online]. Available: https://github.com/yunchih/ORB-SLAM2-GPU2016-final
[4] "Mini-ERA: Simplified version of the main ERA workload," https://github.com/IBM/mini-era, 2021.
[5] S. Arda, A. NK, A. Goksoy, N. Kumbhare, J. Mack, A. Sartor, A. Akoglu, R. Marculescu, and U. Ogras, "DS3: A system-level domain-specific system-on-chip simulation framework," 2020.
[6] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," in *European Conference on Parallel Processing*. Springer, 2009, pp. 863–874.
[7] Barcelona Supercomputing Center, "Nanos++," https://pm.bsc.es/nanox.
[8] ——, "Nanos++ runtime library," https://github.com/bsc-pm/nanox, 2021.
[9] T. Begin and A. Brandwajn, "A note on the accuracy of several existing approximations for M/Ph/m queues," in *4th IEEE International Workshop on High-Speed Network and Computing Environment*, ser. HSNCE 2013, Jul. 2013, p. 5.
[10] H. Darweesh, E. Takeuchi, K. Takeda, Y. Ninomiya, A. Sujiwo, L. Y. Morales, N. Akai, T. Tomizawa, and S. Kato, "Open source integrated planner for autonomous navigation in highly dynamic environments," *Journal of Robotics and Mechatronics*, vol. 29, no. 4, pp. 668–684, 2017.
[11] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.
[12] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.
[13] V. Gupta, M. Harchol-Balter, J. Dai, and B. Zwart, "On the inapproximability of M/G/K: Why two moments of job size distribution are not enough," *Queueing Systems*, vol. 64, pp. 5–48, 08 2010.
[14] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 749–765.
[15] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.
[16] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.
[17] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "A case for guarded power gating for multi-core processors," in *IEEE 17th International Symposium on High Performance Computer Architecture*, ser. HPCA 2011, 2011, pp. 291–300.
[18] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
[19] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.
[20] A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. Vega, C. Villavieja, Y. Etsion, and A. Ramirez, "Scalable simulation of decoupled accelerator architectures," *Universitat Politecnica de Catalunya, Tech. Rep. UPCDAC-RR-2010-14*, 2010.
[21] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 7, pp. 1442–1468, 2013.
[22] H. Topcuoglu, S. Hariri, and M.-y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE transactions on parallel and distributed systems*, vol. 13, no. 3, pp. 260–274, 2002.
[23] A. Vega, J.-D. Wellman, and A. Amarnath, "Scheduling techniques optimization in heterogeneous multi-processors," https://github.com/IBM/stomp, 2021.
[24] J.-D. Wellman, A. Vega, and A. Amarnath, "Scheduler library (SL)," https://github.com/IBM/scheduler-library, 2021.
[25] Wikipedia contributors, "Kendall's notation — Wikipedia, the free encyclopedia," https://en.wikipedia.org/wiki/Kendall%27s_notation, 2020, [Online; accessed 25-June-2020].
[26] G. Xie, G. Zeng, Z. Li, R. Li, and K. Li, "Adaptive dynamic scheduling on multifunctional mixed-criticality automotive cyber-physical systems," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 8, pp. 6676–6692, 2017.
[27] X.-J. Xu, C.-B. Xiao, G.-Z. Tian, and T. Sun, "Hybrid scheduling deadline-constrained multi-DAGs based on reverse HEFT," in *2016 International Conference on Information System and Artificial Intelligence (ISAI)*. IEEE, 2016, pp. 196–202.