Representations for Scheduling of Heterogeneous Computation to Support Motion Planning

Justin McGowen, Ismet Dagli, Mehmet Belviranli, and Neil Dantam
Department of Comptuter Science, Colorado School of Mines, Golden, CO 80401, USA.
Email: {jmcgowen,ismetdagl,belviranli,ndantam}@mines.edu

I. Introduction

***Abstract*—Heterogeneous (multi-accelerator) chips are becoming commonplace in many robotics applications due to their ability to accelerate multiple types of computational workloads. By effectively using these devices, robotic tasks can be performed faster or with less resource usage. However, robotic systems present a variety of requirements for scheduling on such devices, especially on mobile robots. These include both time-critical requirements based on physical constraints (e.g., computing fast enough to safely handle unexpected obstacles) and resource constraints (e.g., battery life). By considering these limitations while scheduling, it is possible to expand the safe operation area of a robot.**

Additionally, changing environments can modify these constraints. In planning situations, this can then impact the optimal plan, as the physical environment may impact viable schedules. Thus, situations can arise where a longer path uses less resources by allowing more efficient schedules. Currently, there is no general, formalized way to reason about these constraints and the tradeoffs they present. A more formal system of reasoning about these tradeoffs also allows them to be considered in higher level tasks, such as Task and Motion Planning.

To solve this problem, we propose the creation of a structured system, the Constrained Autonomous Workload Scheduler (CAuWS). By using a representative language (AuWL), Timed Petri nets, and mixed-integer linear programming, CAuWS offers novel capabilities to represent heterogeneous computation alongside physical constraints, optimization criteria, and motion planning. This enables robots to optimally leverage new computational platforms. We demonstrate CAuWS with a simulation of a drone running vision tasks under multiple physical constraints, showing CAuWS is practical for dynamic environments and obeys physical constraints.

Systems such as mobile robots or drones have two primary limitations on time, energy, and other resources. The first limitations are physical—e.g., braking time or rotor actuation power. The other limitations are computational—i.e., moving and processing data takes time and energy.

Heterogeneous processors allow us to make tradeoffs between the limitations present on mobile robots. Even without parallellization, heterogeneous devices can achieve a more optimal performance than a single device. Consider the GPU and DLA on Nvidia's Xavier devices [21] - the DLA can run vision networks with less energy, at the cost of computation time. If the lowest energy computation is desired, then the DLA should be chosen. However, the real world places limits on computation time that may force the use of GPU.

Considering scheduling under the physical constraints thus requires an upper bound on values such as time or energy. Finding the optimal schedule (mapping tasks to processors) under such constraints is non-trivial on heterogeneous architectures due to the high number of accelerators with diverse characteristics, and is known to be NP-complete.

This also enables higher level reasoning about task and motion planning. By considering not only how the physical limitations impact the schedules, but also how the chosen path impacts the physical limitations (and thus schedules) the robot encounters, it is possible to find more optimal paths balancing physical and computational resource use.

CAuWS enables a generalized solution to the heterogeneous scheduling problem that accounts for the physical constraints, whose high-level operation is illustrated in Fig. 1. CAuWS is able to assign operations from a wide variety of computational workloads to PUs on heterogeneous architectures, creating a set of static schedules for resource-constrained and time-critical systems. These schedules are optimal with respect to the user-defined objective (including time) and profiling, and despite being static, can account for changing conditions.

To our knowledge, there is currently no such general, formalized way to reason about the physical tradeoffs and limitations for scheduling - and this reasoning is often necessary to get the full benefit of a heterogeneous device. Typical approaches for scheduling are separate from task or motion planning, often neglecting physical constraints entirely and instead seeking to greedily optimize time or energy.

To address this, this paper makes the following contributions:

- Our Autonomous Workload Language (AuWL) supports the specification of data-flow and physical constraints that a schedule must obey for a variety of robotic problems.
- Generating schedules from constraints enables formulation of a set of constraints that combine many concerns—such as safety, speed, or energy tradeoffs—into one problem.
- Timed Petri nets provide a convenient intermediate representation between problem definition and constraints.
- By leveraging existing, highly-engineered constraint solvers—e.g., for Satisfiability Modulo Theories (SMT) and mixed-integer linear programming (MILP) [9, 4] —we can easily and programmatically add additional constraints and optimization criteri.
- The approach is evaluated on a simulated PX4 drone, with operations running on a real NVidia Xavier SoC, showing practicality.

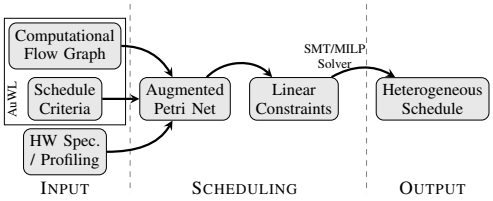

Fig. 1: Overview of CAuWS.

II. Related Work

Scheduling for a variety of computational workloads on multiple-accelerator (heterogeneous) systems has been investigated [5, 18, 19, 24, 27, 1, 12, 15, 22, 2, 25, 8]. Our proposed solution differs by mapping the physical dynamics to the characteristics of heterogeneous computing.

A limited number of studies have structurally approached timing in computation for robots by building models relating physical constraints and computational elements [16, 26, 13].

While a variety of works consider energy in motion planning, only a few works have investigated integrating resource usage from computation into motion planning, such as balancing the energy between computation and movement optimization[23]. This work only considers energy for static motion planning, and not the computation performed during operation.

Petri nets are a form of directed graph. A limited number of studies considered Petri nets for CPU-based, formal scheduling representations for real-time [17, 20, 28] and hybrid [11, 29] systems.

Constraint solving is used in automated planning [14], robotics [7], and program verification and synthesis [3, 10]. Critically, modern, highly-engineered constraint solvers are quite efficient [9, 4]. Previous scheduling techniques have also used constraint solving in an ad-hoc manner.

III. Petri Nets for Scheduling

This section briefly details Petri nets. [1].

A Petri net is a directed, bipartite graph.

Definition 1: A Petri net is the tuple $\mathcal{N} = (P, T, E)$, where,

- P is the finite set of *place* nodes,
- T is the finite set of *transition* nodes,
- $E \subseteq (P \times T \cup T \times P)$ are the *edges* between places and transitions.

Each place P may contain a number of *tokens*. We call the number of tokens contained in all places a *marking*. When a particular transition *fires*, it changes the marking by decrementing the tokens at incoming places and incrementing tokens at outgoing places. Starting and finishing token counts are defined for each place, to define an accepted sequence of firings.

Petri nets make a convenient model for shared resources in parallel systems, as their graph can represent task dependencies, with tokens representing data flow, resource consumption, and processor claiming.

For this paper, we use timed Petri nets, where a transition has a delay between receiving sufficient inputs and firing. We also add edge weights instead of working with integer tokens.

[1]For thourough coverage of Petri nets, we refer the reader to texts such as [6].

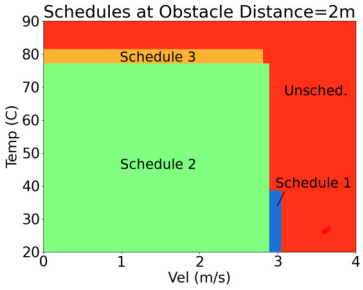

Fig. 2: The set of schedules generated. CAuWS can precompute multiple schedules for varying physical parameters, hadling changing conditions. Schedule 1 is fast but uses more energy scheduling all 3 tasks on all 3 devices simultaneously, while Schedule 3 is the reverse, only using the CPU and DLA.

IV. CAuWS : Proposed Methodology

CAuWS performs several transformations on its input to eventually reach a schedule. **The input** to CAuWS is a specification of (1) *the control flow graph* (CFG) in the AuWL file, (2) the necessary *performance criteria* , and (3) the *profiling data* for estimated running times and energy consumption. **The output** of CAuWS is a schedule with (1) *the set of tasks*, (2) *the ordering of tasks*, and (3) *the mapping of tasks to PUs*.

Our method assumes that the input CFG is static. For a fixed workload including running several vision tasks (which can occur for drones), this is an acceptable assumption. We then introduce the novel Autonomous Workload Language. AuWL describes the data flow of tasks and the necessary schedule criteria. It also includes empirical profiling of task timings and power usage. We use AuWL's specification to construct a timed Petri net intermediate representation which captures the dependency structure, timings, and other quantities. We then generate Mixed-Integer Linear Programming constraints representing its operation that are then solved to obtain a schedule. We use a pre-existing SMT solver (specifically, Z3 [9, 4]) to find a solution the constraints, which corresponds to a sequence of Petri net firings representing a valid schedule.

V. Handling Dynamic Conditions Statically

While it may at first seem that CAuWS is impractical to handle dynamic conditions due to the intensive computation of an NP-complete problem, it is possible to generate a set of schedules that can handle these changing conditions. It is still necessary to know the properties of the robot and computational tasks beforehand. To generate such a set of schedules, we consider the schedule output by CAuWS as a function of input parameters: the physical quantities that vary, such as velocity, obstacle distance, and temperature. For monotonic constraints, this function results in convex schedule regions. We can search for these divisions, resulting in a lookup table such as Fig. 2 that can be used at runtime. This avoids expensive calls to CAuWS during operation.

VI. Experiments

We evaluate CAuWS in two case studies, demonstrating it's adaptability and adherence to constraints.

A. Case Study 1: Environment-limited Search and Rescue

The first simulation is a search-and-rescue task with a quadcopter in a burning house. The drone must explore the house to monitor fires and discover those in need of help by distributing SLAM and 2 vision networks between a CPU, GPU, and DLA. This creates three constraints. Once the corresponding physical equations are expressed in the AuWL file, CAuWS successfully schedules for constraints. These constraints are:

- Heat: The varying temperatures from fire limit computation to avoid overheating.
- Latency: As the compute latency increases, the stopping distance increases limiting "reaction" time
- Power: The power at any given moment is shared between the rotors and the computation, limiting velocity.

In this study, an autonomous drone, simulated with physics, follows a predefined route through a hallway corner with a heat source. The drone's travel over the hallway results in ambient temperatures ranging from 27 to 73 $^{\circ}C$, velocities up to 9.73$\frac{m}{s}$, and obstacle distances as low as 0.3m. A trace of this simulation is taken by recording temperature, velocity, and obstacle distance. These values were then fed into CAuWS, which successfully finds optimal schedules for the simulation.

CAuWS results in the set of schedules shown in Fig. 2. Theoretically, safe schedules can be found as long as the drone remains in the scheduleable reasons. These schedules were then ran on a Xavier AGX to follow the simulation trace, with each schedule running multiple times in a row over the course of the drone's path. This resulted in a total execution time of 9.96s and energy of 84.5J, compared to the predicted time of 9.19s and energy of 96.7J. These errors are reasonable given a safety margin, and could likely be resolved with profiling that addressed caching, contention, and other effects when the operations are not isolated.

B. Comparison with the state-of-the-art

Most multi-accelerator scheduling algorithms do not integrate physical constraints into scheduling decisions, which would result in violation of the constraints, either not using faster operations around the corner, or using too power-intensive operations and overheating.

To the best of our knowledge, *Sky is not the limit* [16] is the closest work that proposes a methodology that integrates a limited set of physical constraints into scheduling decisions. To compare CAuWS against *F-1*, we show that CAuWS results in equivalent or better decisions in 11 different experiment setups.

C. Case Study 2: Discovery & Tracking

We also demonstrate CAuWS's ability to map physical constraints of robots to computational scheduling with two additional simulations, where an aerial drone must discover and follow another aerial adversary.

1) Criteria 1: Power Limits

Drone power is shared between computation and the actuation (rotors), and the total power a battery can provide is limited. When rotor power is too high, computation power must drop. Fig. 3 is a timeline of resulting power consumption.

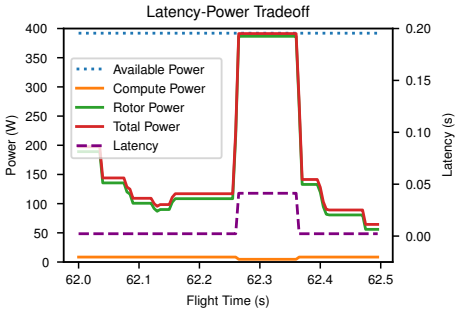

Fig. 3: Power consumption in a pursuit flight. CAuWS balances power and latency under total power limits. Power values are the max over a .05s window to conservatively satisfy power limits.

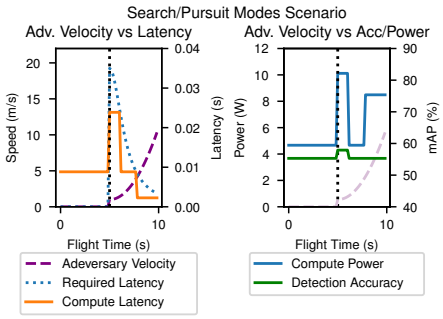

Fig. 4: CAuWS can create schedules covering multiple operation "modes" in a simulated flight, enforcing different constraints for each.

2) Criteria 2: Multiple Modes and Latency Limits

The drone discovery and tracking scenario consists of two modes: looking for an adversary and following the adversary. These two modes impose different scheduling criteria. Having multiple modes expands the range of dynamic situations we can account for with a pre-generated set of schedules.

Fig. 4 shows the results of this multi-mode scenario. CAuWS successfully chooses the lowest power option leading up to the encounter, and satisfies the constraints in desired order during pursuit.

VII. Conclusion

We presented *CAuWS*, a system to represent and generate schedules that satisfy physical and computational constraints. CAuWS can refine the safety margin of robot operation, and supports further work in energy-optimal motion planning. sWe specify computational workloads and schedule criteria with the novel AuWL representation, construct augmented Petri nets, and generate and solve constraints to find the schedule. We validate the generated schedules satisfy constraints and adapt to changing physical conditions with a simulated drone. *CAuWS* offers foundation to address an expanded set of scheduling and planning problems that satisfy physical constraints on heterogeneous platforms.

References

[1] H. Arabnejad and J. G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *IEEE TPDS*, 2014.

[2] Riyadh Baghdadi, Jessica Ray, Malek Ben Romdhane, Emanuele Del Sozzo, Abdurrahman Akkas, Yunming Zhang, Patricia Suriana, Shoaib Kamil, and Saman Amarasinghe. Tiramisu: A polyhedral compiler for expressing fast and portable code. In *CGO*, 2019.

[3] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Advances in Computers*, 58, 2003.

[4] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz-an optimizing SMT solver. In *TACAS*, volume 15, pages 194–199, 2015.

[5] Kevin J Brown, Arvind K Sujeeth, Hyouk Joong Lee, Tiark Rompf, Hassan Chafi, Martin Odersky, and Kunle Olukotun. A heterogeneous parallel framework for domain-specific languages. In *PACT*, 2011.

[6] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*, volume 2. Springer, 2008.

[7] Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. An incremental constraint-based framework for task and motion planning. *IJRR*, 37(10): 1134–1151, 2018.

[8] Mohammad I Daoud and Nawwaf Kharma. A hybrid heuristic–genetic algorithm for task scheduling in heterogeneous processor networks. *JPDC*, 2011.

[9] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[10] Leonardo De Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. *Communications of the ACM*, 54(9):69–77, 2011.

[11] I. Demongodin and N. T. Koussoulas. Differential petri nets: representing continuous systems. *IEEE Trans. on Automatic Control*, 1998.

[12] Dominik Grewe and Michael FP O'Boyle. A static task partitioning approach for heterogeneous systems using opencl. In *Intl. conference on compiler construction*, pages 286–305. Springer, 2011.

[13] Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. Quantifying the design-space tradeoffs in autonomous drones. In *ASPLOS*, pages 661–673, 2021.

[14] Henry A. Kautz and Bart Selman. Planning as satisfiability. *ECAI*, 92:359–363, 1992.

[15] Minhaj Ahmad Khan. Scheduling for heterogeneous systems using constrained critical paths. *Parallel Computing*, 38(4-5):175–193, 2012.

[16] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Paul Whatmough, Aleksandra Faust, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. The sky is not the limit: A visual performance model for cyber-physical co-design in autonomous machines. *IEEE CAL*, 2020.

[17] Sekhri Larbi and Slimane Mohamed. Modeling the scheduling problem of identical parallel machines with load balancing by time petri nets. *Intl. Journal of Intelligent Systems & Applications*, 7(1), 2014.

[18] K. Li, X. Tang, and K. Li. Energy-efficient stochastic task scheduling on heterogeneous computing systems. *IEEE Trans. on Parallel and Distributed Systems*, 25(11): 2867–2876, 2014. doi: 10.1109/TPDS.2013.270.

[19] X. Mei, X. Chu, H. Liu, Y. Leung, and Z. Li. Energy efficient real-time task scheduling on cpu-gpu hybrid clusters. In *INFOCOM*, 2017.

[20] Martin Naedele. Petri net models for single processor real-time scheduling. *Citeseer*, 1998.

[21] NVIDIA. Ai-powered autonomous machines at scale — nvidia jetson agx xavier. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/. (Accessed on 4/21/2022).

[22] Seren Soner and Can Özturan. Integer programming based heterogeneous cpu–gpu cluster schedulers for slurm resource manager. *Journal of computer and system sciences*, 81(1):38–56, 2015.

[23] Soumya Sudhakar, Sertac Karaman, and Vivienne Sze. Balancing actuation and computing energy in motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4259–4265. IEEE, 2020.

[24] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (pie). In *ISCA*, 2012.

[25] Sven Verdoolaege, Juan Carlos Juega, Albert Cohen, Jose Ignacio Gomez, Christian Tenllado, and Francky Catthoor. Polyhedral parallel code generation for cuda. *TACO*, 2013.

[26] Zishen Wan, Aqeel Anwar, Yu-Shun Hsiao, Tianyu Jia, Vijay Janapa Reddi, and Arijit Raychowdhury. Analyzing and improving fault tolerance of learning-based navigation systems. In *DAC*, 2021.

[27] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core. In *PACT*, 2010.

[28] Haitao Zhang and Feiyue Wang. A review of petri net based modeling and verification for embedded real-time systems. In *IDETC-CIE*, 2005.

[29] Zhigang Hu, Xiangtao Jiang, and Jianbiao He. Performance analysis technique for fixed priority scheduling with hybrid real-time transactions. In *2008 Intl. Conference on Information and Automation*, pages 509–513, 2008. doi: 10.1109/ICINFA.2008.4608053.