Brief Industry Paper: Enabling Level-4 Autonomous Driving on a Single $1k Off-the-Shelf Card

Hsin-Hsuan Sung*, Yuanchao Xu*, Jiexiong Guan†, Wei Niu†,
Bin Ren†, Yanzhi Wang‡, Shaoshan Liu§, Xipeng Shen*
North Carolina State University, Raleigh, NC, USA*, College of William and Mary, Williamsburg, VA, USA†,
Northeastern University, Boston, MA, USA‡, PerceptIn, Santa Clara, CA, USA§
*{hsung2, yxu47, xshen5}@ncsu.edu, †{jguan, wniu, bren}@email.wm.edu,
‡yanz.wang@northeastern.edu, §shaoshan.liu@perceptin.io

2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS) | 978-1-6654-9998-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/RTAS54340.2022.00032

***Abstract*—In the past few years we have developed hardware computing systems for commercial autonomous vehicles, but inevitably the high development cost and long turn-around time have been major roadblocks for commercial deployment. Hence we also explored the potential of software optimization. This paper, for the first-time, shows that it is feasible to enable full level-4 autonomous driving workloads on a single off-the-shelf card (Jetson AGX Xavier) for less than $1k, an order of magnitude less than the state-of-the-art systems, while meeting all the requirements of latency. The success comes from the resolution of some important issues shared by existing practices through a series of measures and innovations.**

I. Introduction

Autonomous driving is a thriving sector with great commercial potentials. However, there have been numerous reports indicating that the high cost has been one of the major roadblocks that slow down the development and adoption of autonomous driving in practice [1]. One of the most costly components in an autonomous vehicle is the compute hardware on which the autonomous driving software executes. At present, even partially autonomous (e.g., level-2) driving systems already require either a high-end accelerator box (e.g., NVIDIA Drive [2] or some kind of custom hardware, and the cost of either is no lower than $10k.

Particularly, in the past few years, we have developed hardware computing systems for commercial autonomous vehicles, but inevitably the high development cost and long turn-around time have been major roadblocks for commercial deployment. Hence we decided to turn to software optimization techniques with the belief that we are far from fully exploiting the potential of software optimization. While many companies have developed hardware accelerators for autonomous driving workloads, software optimization has not yet sufficiently explored.

This paper, for the first-time, demonstrates the feasibility of enabling full level-4 autonomous driving workloads on a single off-the-shelf card (Jetson AGX Xavier) for less than $1k, an order of magnitude less in cost and computing power than the autonomous driving hardware used in the current industry, while meeting all the requirements of latency. The success comes from the resolution of some important issues shared by existing practices through a series of measures and innovations.

Specifically, we conduct a focused study to optimize the three deployments of level-4 **A**utonomous **D**riving **App**lications (**ADApp**: derived from Autoware [3], showing the workflow in Figure 1 and the details of the perception components in TABLE I) on a single off-the-shelf low-end card, Jetson AGX Xavier [4] from NVIDIA, as part of the core module of the computing system of the second-generation level-4 autonomous driving system of our company. The result overturns some common perceptions held by the industry. For the first time, we show that it is possible to run industrial-level level-4 autonomous driving on a single off-the-shelf card (Jetson) for as little as $1k while meeting all latency requirements. Meanwhile, this study produces a set of key insights on the important pitfalls or technical deficits in the current autonomous driving industry practice and contributes several practical solutions:

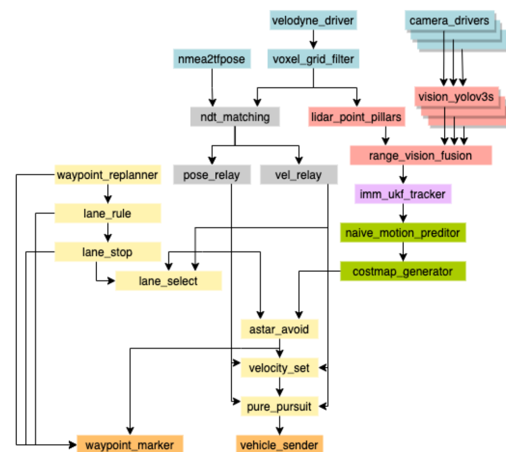

Fig. 1: Workflow of the experimented Autonomous Driving Applications (ADApp). The tasks are categorized into seven modules: Sensing (blue), Perception (red), Localization (gray), Tracking (purple), Prediction (green), Planning (yellow), and final control output (orange)

- Deficit I: Starvation happens when prior scheduling schemes are applied to autonomous driving applications that are deployed to a single low-end device.
  - Resolution: We propose a simple solution, *just-in-time*

978-1-6654-9998-9/22/$31.00 ©2022 IEEE
DOI 10.1109/RTAS54340.2022.00032

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on November 15,2022 at 05:23:56 UTC from IEEE Xplore. Restrictions apply.

TABLE I: The Perception components of ADApp applications

Application	Perception component
ADy288	10 Yolo-V3 models processing ten streams of videos of 288×288 resolution each frame & a 3D point pillar model
ADy416	5 Yolo-V3 models processing five streams of videos of 416×416 resolution each frame & a 3D point pillar model
ADy608	3 Yolo-V3 models processing three streams of videos of 608×608 resolution each frame & a 3D point pillar model

priority adjustment, which resolves the starvation by adjusting the affinity and priorities of tasks in a just-in-time manner.

- Deficit II: Some types of accelerators are left substantially under-utilized due to hardware-oblivious model designs and implementations.
  - Resolution: We employ *hardware-aware model customization*, an approach that significantly increases the accelerators' utilization by bridging the gap between DNN models and multiple types of accelerators.
- Deficit III: Current scheduling algorithms for autonomous driving cannot deal with hybrid workloads that can employ multiple types of accelerators.
  - Resolution: We propose *DAG instantiation based scheduling*, an approach that extends the scheduling of autonomous driving to meet the needs via accelerator-based DAG instantiation.

The explorations together lead to the success of making all of the level-4 autonomous driving applications achieve real-time performance on a single Jetson card. The success has multi-fold implications. It entails the need for the industry and the research community to reexamine some assumptions (on architecture, power budget, cost, the impact of interference, etc.) commonly held on autonomous driving systems, which in turn may lead to a series of new research opportunities, such as (i) reexamining the entire system design in the backdrop of the completely different power and space budget, (ii) adding redundancy and reliability to low-end devices in a cost-effective manner for level-4 autonomous driving, (iii) reconsidering the research on fine-grained scheduling optimizations under the new deployment settings, (iv) reexamining the design, optimization, and deployment of other kinds of autonomous driving applications (e.g., those based on strongly-integrated multi-task DNNs).

II. Key Optimizations

This section presents the three optimizations we developed that are essential for fitting autonomous driving applications on resource-constrained devices.

A. Just-In-Time Affinity and Priority Adjustment

This first optimization addresses a flaw in existing schedulers for autonomous driving. The default Linux time-sharing scheduler is not tailored to autonomous driving. To better meet the realtime requirements of autonomous driving, prior work has developed scheduling algorithms customized to this domain of applications. A representative is ROSCH [5]. Although it showed improvement over the default Linux time-sharing scheduler on the high-end devices [5], ROSCH gave miserable performance on the low-end more constrained Jetson card; as segment 2 of Table II shows, all three ADApps miss all deadlines under ROSCH. They actually work better on the default Linux scheduler, although the 2D perception still misses all deadlines there.

Our analysis reveals that the reason comes from scheduling starvation caused by the priority settings in ROSCH. In ROSCH, every task is put into a real-time queue and has a statically assigned priority as calculated by the HEFT-like algorithm [6]. Each task is a process consisting of multiple threads. All these threads inherit the priority of the process. As a result, the threads of the Sensing node, for their higher priorities, hog the CPU core throughout the execution. The Perception nodes on the same core are starved. As the other modules depend on the output of the Perception module, they make no progress either. The exception is the Planning module. The module always gives outputs in a fixed frequency; as it receives no updates from the other modules, its outputs are out of date and useless. Such a problem does not show up in high-end devices where all main modules can get their dedicated computing units.

Algorithm 1 outlines our improved scheduling algorithm. Changes are made when a work item is put into the input queue of a task, and the main thread of that task is about to be invoked to process that item. At that moment, the main thread of that task is set to SCHED_FIFO, and its priority is changed to p, the level calculated by the default HEFT algorithm [6]. Please note, the changes are made to only the main thread, and the other threads of that task keep the default priority. As soon as the main thread finishes processing the item, it is set back to SCHED_Other, and its priority is reset to the default. In this JIT scheme, the time for a node to hold high priority is reduced, and also the non-critical assistant threads are not promoted to a high priority, which also reduces the core contention.

We only assign real-time priority to the main sub-task in each node, so there is small number of real-time sub-tasks need to be assigned to real-time scheduler. After doing this, the Nvidia AGX Jetson Xavier has enough CPU resources to handle the entry nodes of the Sensing module. Starvation that happens in ROSCH does not happen.

B. Hardware-Aware Model Customization

Our second optimization transforms the DNN models to make them better utilize the accelerators, particularly the Deep Learning Accelerators (DLAs) on Jetson. DLAs are special deep learning accelerators designed to accelerate some common operations in DNNs. For a DNN to take advantage of DLAs, however, the applications must be written with TensorRT in some required form. TensorRT [7] is a library developed by NVIDIA for faster DNN inference on NVIDIA devices. Although current autonomous driving applications (e.g., Autoware) try to take advantage of accelerators, they

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on November 15,2022 at 05:23:56 UTC from IEEE Xplore. Restrictions apply.

TABLE II: Execution time (mean± std) of each module in the ADApp applications on Jetson AGX Xavier and the miss rates. The ∞ represents timeout. The `miss rate` of a module is how often the module misses its expected latency (shown in the parentheses in the table header)—up to 10% over is allowed to tolerate system noises. The column `Miss Rate` shows the miss rates of the most sluggish modules (whose times are prefixed with an $*$), that is, the modules with the largest miss rate in the application.

Application	Running Time of Each Module (ms) **[expected latency in brakets]**							Miss Rate
	Sensing [100ms]	*3D Percept* [100ms]	*2D Percept* [100ms]	*Localization* [100ms]	*Tracking* [100ms]	*Prediction* [100ms]	*Planning* [10ms]	
1. Default Linux Time Sharing								
ADy288	14.3 ± 5.2	94.7 ± 12.8	$*193.3 \pm 17.5$	89.5 ± 30.5	0.9 ± 0.8	0.4 ± 1.0	1.0 ± 0.1	100%
ADy416	15.3 ± 5.1	90.2 ± 12.0	$*167.6 \pm 12.7$	89.1 ± 29.1	0.9 ± 0.7	0.5 ± 0.9	1.1 ± 0.2	100%
ADy608	14.8 ± 4.8	89.0 ± 18.7	$*192.8 \pm 16.2$	91.5 ± 31.2	1.1 ± 0.7	0.4 ± 0.9	1.1 ± 0.2	100%
2. Default ROSCH								
ADy288	8.8 ± 1.0	$*\infty$	$*\infty$	$*\infty$	$*\infty$	$*\infty$	1.1 ± 0.8	100%
ADy416	8.5 ± 0.7	$*\infty$	$*\infty$	$*\infty$	$*\infty$	$*\infty$	1.3 ± 0.9	100%
ADy608	8.5 ± 0.8	$*\infty$	$*\infty$	$*\infty$	$*\infty$	$*\infty$	1.1 ± 0.7	100%
3. JIT Adjustment + DAG Instantiation-Based Scheduling								
ADy288	8.5 ± 0.9	94.6 ± 13.4	$*194.7 \pm 16.3$	43.5 ± 10.2	1.0 ± 0.7	0.6 ± 1.1	1.2 ± 0.4	100%
ADy416	8.4 ± 1.0	91.7 ± 11.2	$*166.8 \pm 11.4$	45.3 ± 11.3	0.8 ± 1.0	0.6 ± 1.1	1.0 ± 0.4	100%
ADy608	8.7 ± 0.7	88.9 ± 17.6	$*190.9 \pm 17.9$	47.2 ± 9.9	1.1 ± 0.6	0.5 ± 0.9	1.0 ± 0.5	100%
4. JIT Adjustment + Hardware-Aware Model Customization + DAG Instantiation-Based Scheduling								
ADy288	8.4 ± 1.2	$*89.0 \pm 15.3$	95.6 ± 5.1	46.3 ± 9.8	0.9 ± 0.9	0.7 ± 0.9	1.0 ± 0.4	0%
ADy416	9.0 ± 0.9	72.0 ± 9.0	88.1 ± 4.3	44.9 ± 10.7	1.0 ± 0.8	0.6 ± 0.9	1.3 ± 0.2	0%
ADy608	8.8 ± 1.2	80.8 ± 10.6	$*98.1 \pm 5.0$	46.4 ± 11.0	1.0 ± 0.7	0.4 ± 1.1	1.1 ± 0.3	0%

Algorithm 1 Just-in-time HEFT algorithm

1: Set the computation costs of tasks and communication costs of edges with mean values
2: Compare $rank_n$ for all tasks by traversing graph upward, starting from the exit task.
3: Sort the tasks in a scheduling list by non-increasing order of $rank_n$ values.
4: **Warp main sub-task in each task by real-time priority adjustment.**
5: **while** there are unscheduled tasks in the list **do**
6:     **for** each processor p_k in the processor-set $p_k \in Q$ **do**
7:         Compute $EFT(n_i, p_k)$ value using the *insection-based scheduling* policy
8:         Assign **main sub-task in task** n_i to the processor p_j that minimizes EFT of task n_i.
9:         **Setup static priority for each wrapped sub-task in each processor according to their EFT in decreasing order.**
10:     **end for**
11: **end while**

often fall in short for missing some subtle properties of the accelerators.

For the three ADApps, for example, our studies show that using DLAs, they actually run slower than without using DLAs on the Jetson card. More specifically, the execution times of the YOLOv3 models and PointPillars both increase. Our analysis shows that the reason is that most of layers in the YOLOv3 could not be supported by DLAs. Those unsupported layers will fall back to GPU to run. This causes not only more time overhead on YOLOv3, but also increases the interference to the executions of PointPillars on GPUs. We augmented the DNN models in ADApps based on the limitations of the DLAs on Jetson. More specifically, we identified the DNN layers in the DNN models that are not supported by TensorRT (the API used to program DLAs) and then replaced those unsupported layers with those that TensorRT supports and provide similar functionalities. One example is to replace the *LeakyReLU* activation functions in every convolution block in the Yolov3 and Yolov3-SPP models with the standard *Relu* activation function. After the accelerator-conscious model augmentation, we retrained the DNN models to ensure that the models give the same accuracy as the original models do. Our experiments show that the accuracy differences are within ±0.015.

Algorithm 2 DAG Instantiation-Based Scheduling

1: D: the DAG
2: $A = \{a_i\}$: the set of accelerators of one or more kinds
3: $V = \{v_i\}$: the set of task nodes in D
4: $E = \{e_i\}$: the set of edges in the D
5: $b_{i,j}$: 1 if v_i can run on a_j, 0 if v_i cannot run on a_j
6: s: a valid assignment from V to A, that is, $\{b_{i,s(i)}$ = 1 for each $v_i\}$
7: $S = \{s\}$: the set of valid assignments
8: $Measures$ = {}
9: **for** each s in S **do**
10:     $taskPerformance \leftarrow$ measure the performance of tasks under s in the default schedule
11:     d = Instantiate D with $taskPerformance$
12:     sch = $scheduleAlgorithm(d)$
13:     $appPerf \leftarrow$ measure the performance of the application under sch
14:     $Measures.add(sch, appPerf)$
15: **end for**
16: $finalSch$ = $Measures.bestPerf()$

C. DAG Instantiation Based Scheduling

With the accelerators' utilization potential unlocked, the next question is to co-schedule the DNN models to the computing units. The design of ROSCH (and HEFT) assumes every node in the DAG can run on any computing unit, which is not the case for the autonomous driving workload where some nodes can run only on CPU, part of some other nodes can run on GPU or DLA. To address the issue, we extend the DAG scheduling algorithm in the previous section to make it workable for systems with multiple types of accelerators. The pseudo-code is shown in Algorithm 2. The design follows two principles (i) being practical (ii) maximizing the quality of the final schedule. An observation is that as our target is a single low-end device, the number of accelerators is very limited (three in the case of Jetson AGX Xavier). Our design hence favors simplicity and result quality over scalability. The basic idea is to enumerate all the possible viable assignments of the task nodes to the accelerators. For each assignment, we instantiate the DAG with the measured performance of the tasks and the communication cost, and then run the

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on November 15,2022 at 05:23:56 UTC from IEEE Xplore. Restrictions apply.

scheduling algorithm in the previous section on that DAG to obtain a schedule. In the end, the schedule that gives the best performance is chosen. The assignment corresponding to the DAG gives the final assignments of the task nodes to the accelerators.

III. Results and Implications

Segments 3 and 4 in Table II report the performance of the three autonomous driving applications after the first or all three optimizations are applied. The JIT priority adjustment eliminates the starvation problem of the default ROSCH scheduling, while the other two optimizations significantly reduce the execution time of the bottleneck, the 2D perception. After the application of all the optimizations, all the modules in the applications can now complete their work within the expected latency (As the table caption marks, 10% over the expected latency is tolerable as such a slack allows real systems to tolerate random fluctuations caused by system noise in real executions.) The applications meet the real-time requirements entirely. Compared to the results in Segment 1, the 3D perception sees about 1.5× time reduction, and the 2D perception sees 2-2.2× time reduction.

It is worth noting that besides the real-time performance requirement, a level-4 autonomous driving system must have enough redundancy, reliability and security to ensure safety. Our autonomous driving system is a full system, the design of which consists of the implementations of security and redundancy (both heterogeneous and homogeneous approaches) besides the core module presented in the paper. Its software stack consists of a middleware to facilitate communications between nodes, as well as a Linux operating system to provide basic system services. This paper intentionally focuses the presentation of the core module because it dominates the computing resource usage and execution time. Our proprietary middleware, for instance, is based on highly optimized nanomsg[1], incurring less than 3% of CPU overheads and communication latency.

Besides pointing out a promising path for the industry to drastically reduce the cost and power of autonomous driving systems, the overturning of the common perceptions by this work also suggests some new research opportunities. Some examples are as follows:

- Architecture design: As the cost and power consumption drop dramatically, it would be valuable to reexamine the design of the entire autonomous driving architecture in terms of the budget allocation for various components, reinforcement of security or reliability, and so on.
- Software design: The changed assumptions on the cost and computing resource suggests the need for rethinking the design of the autonomous driving software, such as the complexities and structures of adoptable DNNs, the inter-component communication, synchronizations, scheduling, and so on.
- Optimizations: There have been lots of research on the optimization of certain points in autonomous driving. They may be worth reexamination. For instance, as everything now runs on a single card, inter-card communication becomes less important, but how to effectively improve the reliability of the low-end device becomes more important. The many optimizations proposed before may work differently in this single-card setting. The research to schedule each layer of a DNN on GPUs to strike an accuracy-power-speed tradeoff [8], for example, may now need to consider the (core, data path, and memory) contentions from other DNN models running on both GPUs and other accelerators (e.g., DLAs).

[1] https://nanomsg.org/documentation.html

IV. Conclusion

With the belief that the potential of software optimization is far from being fully exploited, in this paper, we present our practical experiences of enabling full level-4 autonomous driving workloads, through software optimization only, to achieve real-time performance on a single low-end card at a cost an order of magnitude lower than the industry norm. We achieved this by addressing three major deficits in the current practices through several practical solutions, including *just-in-time affinity and priority adjustment*, *hardware-aware model customization*, and *DAG instantiation based scheduling*. This work overturns the assumption that hardware acceleration is required to achieve better performance and energy efficiency, and serves as a foundation, especially for the autonomous driving industry, of a promising path for drastically lowering the cost and power consumption of commercial autonomous vehicles.

References

[1] S. Liu and J.-L. Gaudiot, "Autonomous vehicles lite self-driving technologies should start small, go slow," *IEEE Spectrum*, vol. 57, no. 3, pp. 36–49, 2020.

[2] "Nvidia drive - autonomous vehicle development platforms," May 2021. [Online]. Available: https://developer.nvidia.com/drive

[3] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[4] 2021. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[5] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "Rosch: Real-time scheduling framework for ros," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2018, pp. 52–58.

[6] 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.

[7] 2021. [Online]. Available: https://docs.nvidia.com/deeplearning/tensorrt /developer-guide/index.html

[8] S. Bateni and C. Liu, "Neuos: A latency-predictable multi-dimensional optimization framework for dnn-driven autonomous systems," in *2020 {USENIX} Annual Technical Conference ({USENIX}{ATC} 20)*, 2020, pp. 371–385.

Authorized licensed use limited to: N.C. State University Libraries - Acquisitions & Discovery S. Downloaded on November 15,2022 at 05:23:56 UTC from IEEE Xplore. Restrictions apply.