Profiling Concurrent Vision Inference Workloads on NVIDIA Jetson

Abhinaba Chakraborty*,Wouter Tavernier*,Akis Kourtis†,Mario Pickavet*,Andreas Oikonomakis†,Didier Colle*
*IDlab Department of Information Technology Ghent University-IMEC, Ghent, Belgium †NCSR Demokritos, Greece

***Abstract*—The necessity of processing real-time data at the network edge is growing. Low-power AI accelerators, especially edge GPUs, help meet this demand by mitigating cloud-related latency and bandwidth issues. However, GPUs remain underutilised, even in heavy workloads, due to a limited understanding of resource sharing in edge computing. This work analyses key GPU metrics: utilisation, memory, streaming multiprocessors (SMs), and tensorcores on NVIDIA Jetson devices under concurrent vision-inference workloads. Our findings show that while GPU utilisation can reach 100% with optimisations, SMs and tensor cores often run at only 15–30% capacity.**

I. Introduction

Deep Neural Networks (DNNs) [1] are widely used in edge AI applications [2], [3]. Traditionally, cloud offloading [4] has been the preferred approach, but inference-on-edge [5], [6] is gaining traction due to its lower latency, bandwidth usage, and power consumption. Advances in model redesign and custom architectures are improving edge-based inference, but hardware and software diversity complicates optimisation. Although libraries like $BLAS$ [7] support deep learning computations, manual hardware-specific tuning is complex. Deep learning compilers and SDKs [8]–[12] automate optimisation through layer and operator fusion techniques. GPUs are widely used for edge inference due to their broad support. Maximising their efficiency requires concurrent execution [13], [14], often managed through virtualisation. Cloud GPUs primarily use time multiplexing [15], which degrades performance with more applications [16], or spatial multiplexing via NVIDIA's Multi-Process Service (MPS) [17], [18]. Newer architectures like Turing [19] and Ampere [20] enhance isolation, but Jetson GPUs [21], [22] lack MPS support, relying on space or time multiplexing. Their unified memory systems reduce CPU-GPU communication overhead but increase memory consumption with concurrent processes. Understanding system capabilities is essential for designing efficient inference systems. High-level tools like NVIDIA Triton Server [23] provide performance insights but lack detailed GPU component analysis. Prior studies examined Jetson devices under vision workloads [24]–[27], while others explored multi-tenancy [28] and micro-architectural performance [29]. However, research has largely focused on high-level analysis using lightweight profiling tools. This study provides a detailed evaluation of Jetson devices, analysing both high- and low-level GPU metrics such as SM and tensor core utilisation under concurrent workloads. Identifying bottlenecks at these levels is crucial for optimising performance, improving hardware efficiency, and enhancing fault tolerance in edge AI applications.

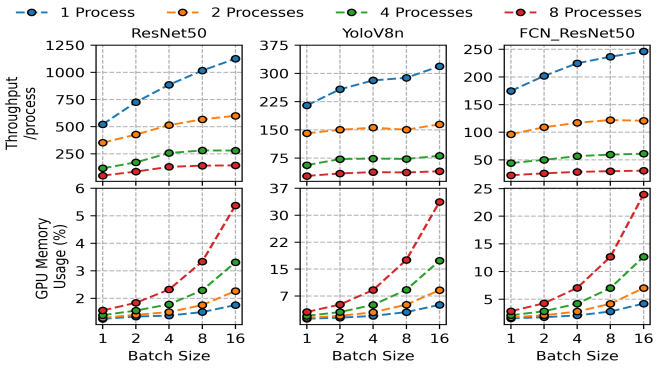

Fig. 1: GPU Memory Usage (%) and T/P for *int8*,ResNet50, FCN_ResNet50, and YoloV8n models on Jetson Orin Nano

II. Experimental Setup

We conduct experiments on two NVIDIA Jetson GPUs: the Orin Nano [20] and the Nano [30]. The Orin Nano leverages Tensor Cores, while the Nano serves as an entry-level option. We used *trtexec* [31] for the execution of the TensorRT models and *Jetson-Stats* [32], *Nsight Systems* [33] for profiling. Our vision workload focuses on image classification with *ResNet50* [34], segmentation with *FCN_ResNet50* [35], and object detection with *YoloV8n* [36] from PyTorch Hub [37]. These models are obtained from the PyTorch hub and then converted to TensorRT models. Due to space constraints, we present results for the Orin Nano, with similar trends observed on the Nano.

III. Workload Analysis

Throughput & GPU Memory Usage In a single-process setup, GPU memory usage in execution runtime depends on the model size plus twice the batch size because *trtexec* pre-enqueues one batch. Each process allocates its memory for multi-process systems, so the memory consumption for each process is the number of processes multiplied by the sum of the model size and twice the batch size. This means that while increasing the batch size can improve throughput per process, adding more concurrent processes eventually reduces the throughput per process due to memory constraints and resource contention. *ResNet50* $int8$ model shows an increase in throughput per process from 500 at a batch size of 1 to 1000 at a batch size of 8, but when scaling to 8 processes, the throughput per process drops to nearly 200 (Fig-1).

Power Consumption Power consumption varies with precision. For *Orin Nano*, $int8$ models use less energy while

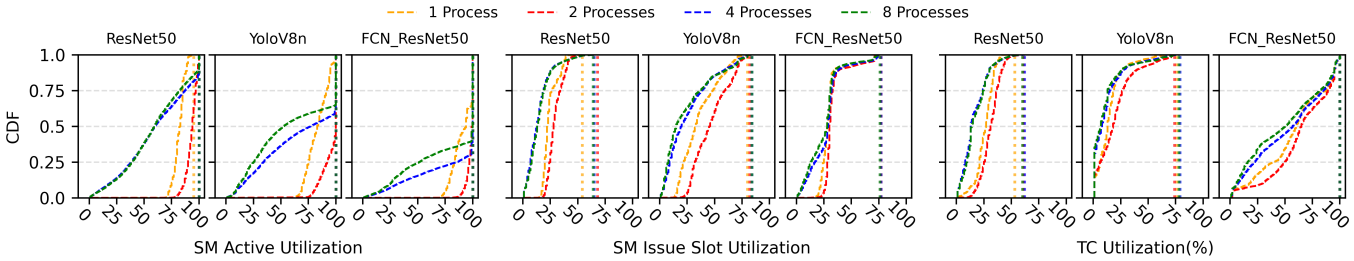

Fig. 2: SM Active and Issue Slot & TC Utilization vs Concurrent Processes

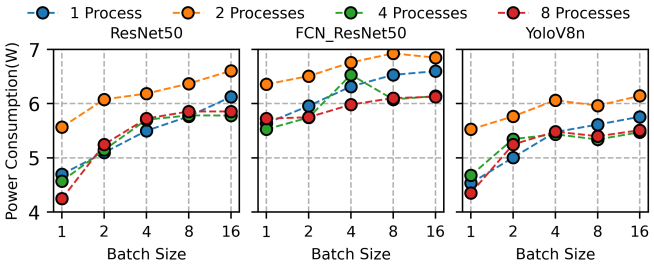

Fig. 3: Power Consumption for *int8 ResNet50*, *FCN_ResNet50* and *Yolov8n* model on Jetson Orin Nano

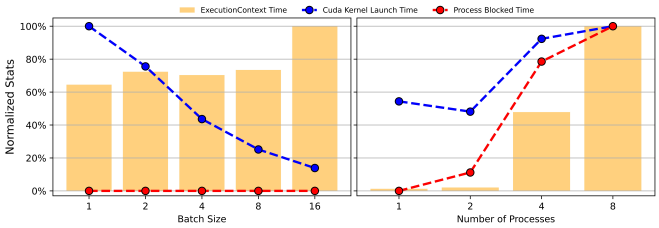

Fig. 4: Comparison of CPU and GPU Events for *ResNet50* $int8$ model on Jetson Orin Nano: vs. Batch Sizes (Left), vs. Process Counts (Right)

achieving the highest throughput. In Fig 3, power consumption appears inconsistent across batch sizes and process counts. At first glance, it increases with batch size; for example, in the 1-process scenario, the power rises from batch size 1 to 16. However, this increase is not smooth, as power levels eventually plateau. This happens because of Dynamic Device and Frequency Scaling (DVFS), a mechanism that reduces the frequency of the GPU to reduce power consumption, which also limits throughput.

SM, Issue Slot and Tensor Core Utilisation We observe that, *ResNet50* achieves 75–90% SM active utilization across all precisions, with *tf32* reaching 100% for 15% of the runtime. However, SM issue slot utilization remains low, mostly between 25 and 40%, indicating issue stalls. *ResNet50* shows a steep CDF, with TC utilization below 50% in *int8* and higher in *fp16* and *tf32*. However, higher TC usage does not always translate to higher throughput, suggesting that other factors, such as memory bandwidth, play a role. In Fig. 2, we compare single batch and multiprocess scenarios, analysing active SM utilisation and issue slot. For *ResNet50*, SM utilisation is mostly 80–100% for 1 and 2 processes, with steeper cumulative curves than 4 and 8 processes. The 2-process setup often reaches 100% SM utilisation. Issue slot utilisation remains around 25%, never exceeding 80%. Tensor Core (TC) utilisation for *ResNet50* averages 25% for 1–2 processes but drops to 15–20% for 4–8. TC usage never exceeds 50%, and its peak increases with process count. However, higher TC utilisation does not always improve throughput due to issue stalls, which intensify with more processes.

IV. Kernel Level Performance Analysis

The execution timeline consists of TensorRT *ExecutionContext* (EC) events interleaved with *Cuda Synchronisation* (CS) events, represented as $\sum_i EC_i + \sum_j CS_j$. With one or two concurrent processes, EC_i duration remains short (1–2 ms), but with four or more, it increases sharply—by nearly $30\times$ and $70\times$ for four and eight processes, respectively. (Fig-4) Batch size has a linear impact on EC_i, enabling higher throughput with only a slight increase in processing time. However, increasing concurrent processes reduces throughput due to scheduling inefficiencies. Each EC_i consists of GPU kernel launches (K_l), CPU thread initiations (T_l), computation time (C_l), and blocking time (B_l).The blocking time results from the *big.LITTLE* [38] architecture, where heavy workloads are assigned to a limited number of CPU cores—three on the Jetson Orin Nano and two on the Jetson Nano.

On the Jetson Orin Nano, when four or more processes run concurrently, time-sharing mode leads to preemption, increasing process blocking time ($\sum_l b_l$), CPU thread rescheduling time ($\sum_l T_l$), and GPU kernel launch overhead ($\sum_l K_l$). This also raises cache miss rates at $L1$ and $L2$ levels, extending computation time ($\sum_l C_l$) and worsening execution efficiency. Similar trends are observed on the Jetson Nano.

V. Conclusions

These findings show the role of CPU-GPU scheduling in shaping GPU performance for inference workloads. They highlight the need for further research into advanced thread scheduling techniques and programmable architectures designed to enhance deep learning acceleration.

Acknowledgment

The research work presented in this article has been supported by the European Commission under the Horizon Europe Programme and the OASEES project (no. 101092702).

REFERENCES

[1] M. Awad *et al.*, *Deep Neural Networks*. Berkeley, CA: Apress, 2015, pp. 127–147. [Online]. Available: https://doi.org/10.1007/978-1-4302-5990-9_7

[2] I. Goodfellow *et al.*, *Deep Learning*. The MIT Press, 2016.

[3] S. Dasiopoulou *et al.*, "Knowledge-assisted semantic video object detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 10, pp. 1210–1224, 2005.

[4] J. Wang *et al.*, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, Feb. 2019. [Online]. Available: https://doi.org/10.1145/3284387

[5] E. Kristiani *et al.*, "Optimization of deep learning inference on edge devices," in *2020 International Conference on Pervasive Artificial Intelligence (ICPAI)*, 2020, pp. 264–267.

[6] J. Fang *et al.*, "Large language models (llms) inference offloading and resource allocation in cloud-edge networks: An active inference approach," in *2023 IEEE 98th Vehicular Technology Conference (VTC2023-Fall)*, 2023, pp. 1–5.

[7] C. L. Lawson *et al.*, "Basic linear algebra subprograms for fortran usage," *ACM Trans. Math. Softw.*, vol. 5, no. 3, p. 308–323, Sep. 1979. [Online]. Available: https://doi.org/10.1145/355841.355847

[8] T. Chen *et al.*, "Tvm: an automated end-to-end optimizing compiler for deep learning," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 579–594.

[9] N. Rotem *et al.*, "Glow: Graph lowering compiler techniques for neural networks," 05 2018.

[10] F. Boemer *et al.*, "ngraph-he: a graph compiler for deep learning on homomorphically encrypted data," ser. CF '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 3–13. [Online]. Available: https://doi.org/10.1145/3310273.3323047

[11] A. Artemev *et al.*, "Memory safe computations with xla compiler," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.

[12] "Tensorrt sdk," https://developer.nvidia.com/tensorrt.

[13] A. A. Süzen *et al.*, "Benchmark analysis of jetson tx2, jetson nano and raspberry pi using deep-cnn," in *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, 2020, pp. 1–5.

[14] B. Fang *et al.*, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 115–127. [Online]. Available: https://doi.org/10.1145/3241539.3241559

[15] Y. Suzuki *et al.*, "Gpuvm: Gpu virtualization at the hypervisor," *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2752–2766, 2016.

[16] R. Ausavarungnirun *et al.*, "Mask: Redesigning the gpu memory hierarchy to support multi-application concurrency," *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:3985258

[17] H. Wu *et al.*, "Safe process quitting for gpu multi-process service (mps)," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 1169–1170.

[18] Y. Peng *et al.*, "Contrast and analysis about the characteristics of mps and cdp in gpu kepler architecture," in *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, 2016, pp. 137–141.

[19] Z. Jia *et al.*, "Dissecting the nvidia turing t4 gpu via microbenchmarking," 03 2019.

[20] H. Abdelkhalik *et al.*, "Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis," 08 2022.

[21] "Nvidia jetson," https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[22] A. Kurniawan, *Introduction to NVIDIA Jetson Nano*. Berkeley, CA: Apress, 2021, pp. 1–6. [Online]. Available: https://doi.org/10.1007/978-1-4842-6452-2_1

[23] "Triton perf analyser," https://docs.nvidia.com/deeplearning/triton-inference-server/.

[24] P. S.K *et al.*, "Characterizing the performance of accelerated jetson edge devices for training deep learning models," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 6, no. 3, Dec. 2022. [Online]. Available: https://doi.org/10.1145/3570604

[25] T. Swaminathan *et al.*, "Benchmarking deep learning models on nvidia jetson nano for real-time systems: An empirical investigation," 06 2024.

[26] H. Halawa *et al.*, "Nvidia jetson platform characterization," 08 2017, pp. 92–105.

[27] S. Valladares *et al.*, *Performance Evaluation of the Nvidia Jetson Nano Through a Real-Time Machine Learning Application*, 01 2021, pp. 343–349.

[28] P. Subedi *et al.*, "Ai multi-tenancy on edge: Concurrent deep learning model executions and dynamic model placements on edge devices," *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pp. 31–42, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236447752

[29] D. Moolchandani *et al.*, "Performance and power prediction for concurrent execution on gpus," *ACM Trans. Archit. Code Optim.*, vol. 19, no. 3, May 2022. [Online]. Available: https://doi.org/10.1145/3522712

[30] "Nvidia maxwell architecture," https://developer.nvidia.com/maxwell-compute-architecture.

[31] "Tensorrt github," https://github.com/NVIDIA/TensorRT/tree/main/samples/trtexec.

[32] "Jetson-stats," https://pypi.org/project/jetson-stats/.

[33] "Nsight systems," https://developer.nvidia.com/nsight-systems.

[34] K. He *et al.*, "Deep residual learning for image recognition," 2016, pp. 770–778.

[35] J. Long *et al.*, "Fully convolutional networks for semantic segmentation," 2015, pp. 3431–3440.

[36] R. Varghese *et al.*, "Yolov8: A novel object detection algorithm with enhanced performance and robustness."

[37] A. Paszke *et al.*, *PyTorch: an imperative style, high-performance deep learning library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[38] "Arm big.little architecture," https://www.arm.com/technologies/big-little.