SVSoC: Speculative Vision Systems-on-a-Chip

Yiming Gan, Yuxian Qiu, Jingwen Leng, Yuhao Zhu *Member, IEEE*

Abstract—Frame latency in continuous vision significantly impacts the agility of intelligent machines that interact with the environment via cameras. However, today's continuous vision systems limit the frame latency due to their fundamental sequential execution model. We propose a speculative execution model along with two mechanisms that enable practical vision speculation. We present SVSoC, a new mobile Systems-on-a-chip (SoC) architecture that augments conventional mobile SoCs with the speculation capability. Under the same energy budget, SVSoC achieves 14.3% to 35.4% latency reduction in different scenarios.

Index Terms—Continuous Vision, Systems-on-a-Chip, Speculation

1 Introduction

Domain specific architectures provide more compute capability with lower energy consumption under the same silicon budget. Among many emerging domains, this paper focuses on *continuous computer vision*, which processes real-time images from camera sensors to extract visual insights that guide high-level decision making. Continuous vision is key to many existing and emerging use-cases such as Augmented Reality and robotics navigation.

A main bottleneck of today's continuous vision systems is their long frame latency, i.e., the latency from when the camera sensor starts sensing the scene to when the vision results are generated. For instance, a typical embedded robot today has a 200 ms responsive latency from event to command, in which 100 ms is attributed to the vision sub-system [4]. Such a high latency puts a hard bound on the agility of the robotic system, degrading user- experience and sometimes presenting great dangers to users [3], [5].

The root-cause of the long frame latency is the *sequential execution model* of today's continuous vision pipeline. Fig. 1 shows that the three main stages in a continuous vision pipeline—sensing, imaging, and vision computation—process a frame sequentially, where the sensing stage produces raw sensor data consumed by the imaging stage, which generates RGB frames consumed by the vision computation stage. Existing system optimizations such as pipelining and batching are designed to improve throughput (frame rate), but further exacerbate the frame latency.

This paper proposes SVSoC, a new mobile Systems-on-a-chip (SoC) architecture that improves the frame latency of continuous vision tasks. The key idea is that we can fundamentally reduce the frame latency by *breaking the sequential execution chain*. Specifically, we propose a new vision execution model where the vision computation stage operates *speculatively* on predicted future frames before the sensing and imaging stages generate the actual frames. Once the actual frame is generated and vindicates the predicted frame, the vision results are likely already available, thus reducing the end-to-end frame latency.

Although the speculative execution model improves frame latency, it presents two major challenges. First, speculation introduces new computations due to frame prediction and thus potentially increases the energy consumption. We leverage the inherent error-tolerant nature of computer vision tasks to approximate the vision computation and thus reduce the amount of computation. Second, the speculative execution model also naturally requires more concurrent computations on the fly, increasing the hardware resource contention. We propose an intelligent scheduling scheme that leverages the heterogeneity available on mobile SoCs to mitigate the resource contention. Our initial results have been promising. SVSoC achieves up to 35.4% frame latency reduction under the same energy budget.

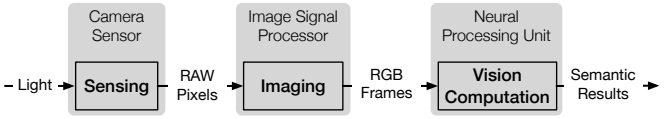


Fig. 1: The conventional sequential execution model of the continuous vision pipeline.

2 Speculative Execution Model

To overcome the limitation of sequential execution model in continuous vision pipeline, we introduce the speculative execution model. We describe the execution model in this section, and discuss its implications.

2.1 Execution Model

The key idea of the speculative execution model is to compute the vision algorithm speculatively on predicted frames before sensing and imaging stages generate actual RGB frames. Once an actual frame is generated, the system uses it to validate the predicted frame. If the predicted and actual frames match by some metric, the vision task results on that frame are likely already available, thus reducing the end-to-end frame latency. Otherwise, the speculated work is discarded and the system starts over with the actual frame.

The speculative execution model is illustrated in Fig. 2 with three frames, in which Frame 1 is executed under the sequential model while Frame 2 and 3 are executed speculatively. Comparing to the conventional sequential model as shown in Fig. 1, the speculative execution model requires two new system components: a frame predictor and a check and commit module (Pred and C&C in Fig. 2, respectively).

The predictor predicts future frames to enable speculation. In particular, we propose to predict multiple frames in a sequence (e.g., Frame 2 and 3 in Fig. 2) to further reduce the frame latency. The C&C module commits vision results on predicted frames only if the predicted frames are checked to match the actual frames. Several image similarity metrics

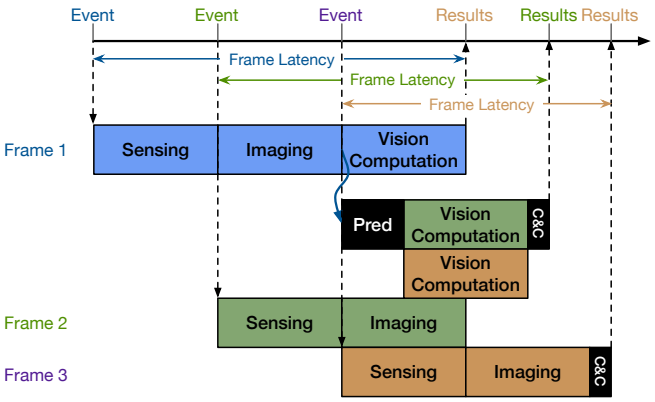


Fig. 2: The speculative execution model. Pred denotes the frame predictor, and C&C denotes "check and commit".

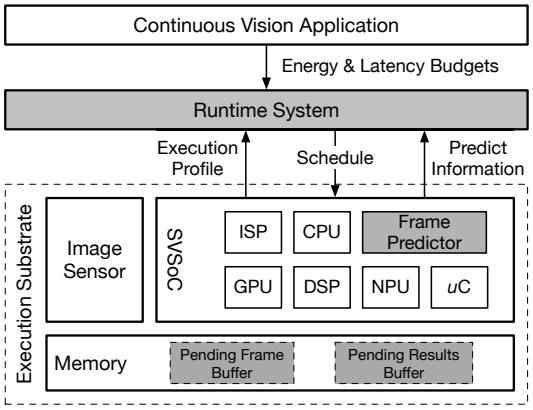


Fig. 3: Overview of a continuous vision system with SVSoC. Agmentations are shaded.

have been proposed in the computer vision literature such as Structural Similarity Index Measure (SSIM) [10] that is lightweight to compute.

2.2 Implications

The speculative execution model has two major inefficiencies that must be addressed in practice. First, it incurs energy overhead due to frame prediction and Check&Commit. Second, speculative execution allows for more outstanding frames to be processed concurrently, which leads to resource contention. For instance, the vision computations of Frame 2 and Frame 3 in Fig. 2 are overlapped owing to speculation. If only one IP block is available to execute vision algorithms, this would serialize the vision stage of the two frames, defeating the purpose of speculation.

3 Enabling Mechanisms

We propose two mechanisms that address the resource contention and energy overhead challenges, respectively, and thus enable practical speculative continuous vision.

Exploiting SoC Heterogeneities To address the resource contention introduced by speculation, we propose to exploit the hardware heterogeneities available on today's mobile SoCs. State-of-the-art mobile SoCs such as Qualcomm Snapdragon 835, HiSilicon Kirin 970, and Apple A11 all provide at least four IP blocks that can be used to execute vision algorithms, including the CPU, GPU, DSP, and NPU.

To leverage the SoC heterogeneity, the outstanding frames will be assigned to different available IP blocks. It is up to the runtime system (discussed in Sec. 4) to decide how to best schedule the frames across different IP blocks.

Front-end Approximation Using other IP blocks alleviates resource contention, but also increases energy consumption. This is because CPUs/GPUs/DSPs are less energy-efficient than NPUs for executing DNNs, aggravating the energy overhead introduces by speculation.

We propose to mitigate the energy overhead through "approximated speculation" where some predicted frames out of the predicted sequence are not checked with the actual frames. For the approximated frames, the sensing and the imaging stages could be switched off, saving up to 50% of the total system power consumption [11] in certain vision use-cases. This form of approximation is different from conventional approximation in computer vision that approximates the vision algorithm; in contrast, we propose to approximate the front-end that generates the frames. This is an opportunity uniquely enabled by speculation, and in turn helps mitigate the speculation overhead.

4 Speculative Vision SoC (SVSoC)

We propose Speculative Vision SoC (SVSoC), which speculatively executes vision algorithms to improve frame latency while leveraging the two enabling mechanisms to reduce energy overhead and to improve hardware utilization. SVSoC is assisted with a runtime layer that intelligently schedules the speculative tasks on the SoC. The runtime system takes the application-level information such as total energy budget and frame latency constraints, and intelligently maps different frames to IP blocks. Fig. 3 shows the speculative continuous vision stack.

Architecture Speculative continuous vision requires two modest augmentations to existing mobile systems, as shown in Fig. 3. First, the frame predictor requires a dedicated IP block so as to not contend the already-scarce hardware resources. As we will show in Sec. 5, frame predictors employ lightweight algorithms that lead to modest additional silicon overhead.

Second, the speculation execution model requires two new data structures: the Pending Frame Buffer that stores predicted frames and the Pending Results Buffer that stores finished results that are not yet committed. For the simplicity of implementation, we reserve two regions in the physical memory for the two data structures.

Under the speculative execution model, the CPU is programmed to manage the speculative tasks on the heterogeneous SoC. The management is performed by a runtime layer as discussed below.

Runtime The speculative execution model exposes a complex scheduling problem to the runtime system where multiple outstanding frames are to be executed on a heterogeneous SoC with multiple execution targets (IP blocks). In particular, the scheduler's goal is to minimize the average frame latency subject to an energy budget. Note that the scheduler could be easily extended to consider the dual problem of minimizing the energy under latency constraint. We consider the former in this paper due to the space limit.

We formulate the scheduling task as a constrained optimization problem. Assuming that each time the frame predictor predicts **M** frames, which are scheduled on the

available IP blocks. The runtime's objective is expressed as:

$$min(\sum_{j=1}^{M} Lat_j/M) \quad s.t.\ E\ < E_{budget} \tag{1}$$

where Lat_j represents frame latency for frame j, and is calculated as the difference between the frame start time (S_j) and the frame finish time (F_j). While S_j can be statically determined as $(j-1) \times T_{sensing}$, F_j depends on whether or not the frame is approximated. An non-approximated frame has to wait until the actual frame is generated and the check is passed, whereas an approximated frame is finished whenever the vision stage finishes.

We find that most prediction algorithms exhibit a general accuracy trend across the predicted frames, which helps us determine which frames to approximate (i.e., to skip checking with real frames). Specifically, from the first a few predicted sequences, the runtime empirically learns the most accurate **K** frames from a sequence of **M** predicted frames. These **K** frames are chosen as the approximate frames for the rest of the execution. We dub **K** "approximate degree", which affects the latency-accuracy trade-off, and we will evaluate its impact later.

This optimization problem is non-linear, which we solve using a greedy algorithm that schedules frames one at a time. Each frame is scheduled to the IP block that would provide the lowest frame latency – provided that the rest of the frames can possibly finish with the remaining energy budget using the lowest energy-consuming IP; otherwise the rest of the frames are scheduled to the lowest energy-consuming IP. The algorithm has a low overhead of about 30 μs running on the Kyro CPU in the Qualcomm Snapdragon 820 mobile SoC (see Sec. 5.2 for experimental setup).

5 Evaluation

We discuss implementation details, introduce our experimental methodology, and evaluate the potential of SVSoC using real-world vision workloads.

5.1 Implementation Details

Our frame predictor adopts a deep recurrent CNN model PredNet [7]. The recurrent model predicts **M** consecutive frames by using both the current frame and the hidden state that contains historical information. Intuitively, predicting more consecutive frames increases the scheduling window, but also introduces a higher chance of mis-prediction as errors accumulate. We empirically find that 10 frames to be desirable, which in turns leads to about 81 million MAC operations, indicating a compute requirement that is over two orders of magnitude lighter than that of a typical vision task such as object tracking and detection [11].

We use the widely-used SSIM metric [10] to assess the quality of the predicted frames with regard to the actual frames. SSIM is lightweight to compute, requiring only 7.4 million MAC operations per frame. The runtime rejects a predicted frame if its SSIM value is below a threshold. We empirically determine the SSIM threshold using object detection as the use-case (see Sec. 5.2 for experimental setup). Fig. 4 shows how the object detection mean average precision (mAP) (y-axis) varies with the SSIM threshold (x-axis) ranging from 0 to 1, where 1 means that a predicted frame must be a pixel-perfect match with the actual frame to be accepted. We choose 0.8 as the threshold because it incurs less than 0.01 mAP loss and a larger value only leads to marginal improvement.

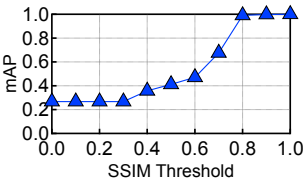


Fig. 4: Accuracy sensitivity to SSIM threshold.

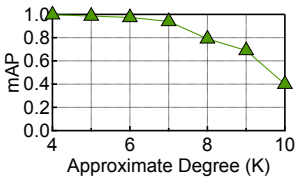
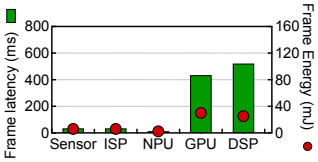
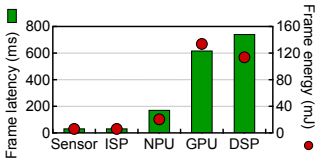


Fig. 5: Accuracy sensitivity to approximation degree.



(a) Front-end-dominate case.



(b) Vision-stage-dominate case.

Fig. 6: Per-frame latency and energy comparisons of IP blocks.

We also empirically determine the approximate degree **K** used by the runtime to control the accuracy loss. Fig. 5 shows that as **K** increases the accuracy drops. For our evaluation, we use a configuration where **K** is 6, corresponding to an accuracy drop less than 0.1. For other use-cases that are more sensitive to accuracy a lower **K** value could be used.

5.2 Experimental Setup

Software setup We use object detection task as a case-study. We use YOLOv2 [8], a state-of-the-art CNN-based detection algorithm, and the widely-adopted KITTI dataset [6]. We evaluate two different input resolutions to mimic two typical continuous-vision scenarios: one with a low resolution (256 $\times$ 128) and thus the front-end (sensing and ISP) dominates the execution time and latency, and the other with a high resolution (608 $\times$ 608) and thus the vision stage dominates the latency and energy.

Hardware setup We model the baseline as a typical mobile SoC consisting of key IP blocks including the CPU, GPU, ISP, and DSP. The image sensor and the ISP operate at 30 FPS for vision-stage-dominate case, and operate at 40 FPS for the front-end-dominate case. In addition, we model an NPU in the baseline. SVSoC uses a specialized IP for the frame predictor. We choose to model the predictor IP as a *separate*, small NPU. We task the CPU with SSIM calculation and runtime scheduling due to their light compute requirements. The size of PFB is 960 KB and 1.1 MB for the two use-cases, respectively, and the PRB size is about 170 Bytes.

We use ScaleSim [9], a RTL-validated DNN accelerator simulator, to model the behaviors of both the main NPU and the predictor NPU. Both NPUs are based on the systolic-array architecture. The main NPU has an array size of 20$\times$20 operating at 500 MHz with a SRAM size of 1.5 MB, and the predictor NPU has an array size of 10 $\times$ 10 operating at 350 MHz with a SRAM size of 128 KB. The area overhead introduced by the predictor NPU is 170,000 um^2, less than 0.15% of the total SoC die area [2].

We use the SoC simulation infrastructure used in Zhu et al. [11], which faithfully mimics the IP interactions and the SoC-memory traffic. When possible, we parameterize the simulator using measurements derived from the Open-Q 820 Hardware Development Kit [1], which contains the Snapdragon 820 SoC that represents a middle-end mobile SoC. Fig. 6a and Fig. 6b show the latencies and energies of

different IP blocks under the two different use-cases.

Baselines We compare with two baselines: 1) `Single-NPU` where only the NPU is used for vision computation, resembling the practice of today's mobile SoCs, and 2) `FCFS` where all the IP blocks are available for vision computation but without speculation capabilities. The runtime uses a First-Come-First-Serve (FCFS) policy which maps an incoming frame to the fastest available IP block based on the order they are generated. `FCFS`'s improvements over `Single-NPU` show the benefits of using multiple IP blocks, and SVSoC's improvements over `FCFS` show the benefits of speculation.

5.3 Results

Front-end-dominant Use-case SVSoC improves the latency significantly in use-cases where the vision stage is relatively simple. Fig. 7a compares the latency reduction of SVSoC with the two baselines under different energy budgets when the input resolution is small. The latency reduction is normalized to the `single-NPU` baseline.

Both `Single-NPU` and `FCFS` require a minimal energy budget of 16.5 mJ/frame to finish executing all the frames–bounded by the total energy consumption of sensing, ISP, and the NPU. Under this energy budget, SVSoC reduces the latency by 35.4%. The latency reduction is mostly contributed by front-end approximation, which allows approximated frames to be committed without waiting for the front-end to finish. The approximation also let SVSoC operate under an energy budget as low as 9.84 mJ per frame.

Further investigations show that SVSoC uses only the NPU because the NPU is not the bottleneck in the front-end-dominate use-case. Our runtime correctly recognizes this and does not schedule the vision stage to other IP blocks.

For comparison purposes, we also show the latency reduction of the oracular SVSoC with an 100% prediction accuracy. Oracle almost completely removes the frame latency while requiring an energy budget of only 5.17 mJ.

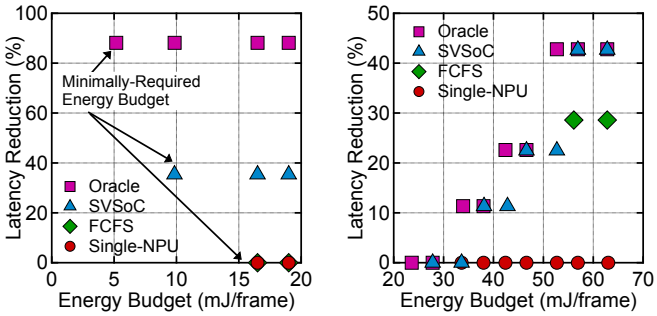
Vision-stage-dominant Use-case In this use case, SVSoC exploits SoC heterogeneities to reduce the frame latency significantly. Similar to Fig. 7a, Fig. 7b compares the latency reduction of different schemes but with higher-resolution images as the input, making the frame latency vision-stage-dominant. `Single-NPU` requires a minimal energy budget of 33.6 mJ per-frame using only the NPU. Under this energy budget, however, SVSoC is not able to make use of other IP blocks because they are more energy-hungry than the NPU (see Fig. 6b). As the energy budget increases to 38.1 mJ, SVSoC leverages non-NPU IP blocks and achieves 11.3% latency reduction compared to `Single-NPU`.

`FCFS` requires a minimal energy budget of 56.1 mJ. In contrast, SVSoC could operate with a much lower minimal energy budget (27.8 mJ) because SVSoC can use the energy saved from switching-off the front-end in the vision stage. Under the same 56.1 mJ budget, SVSoC achieves 14.3% latency reduction compared to `FCFS`. This is because speculation exposes more outstanding frames to the SoC, which could better schedule frames for latency reduction. Finally, our SVSoC design achieves slightly lower latency reductions and slightly higher energy consumption compared to the Oracle, owing to the mis-predictions overhead.

5.4 Discussions on Applicability

Our paper provides a limit study as to how far speculative systems can be applied to real-world applications. The benefits of SVSoC is more significant in use-cases where the front-end dominates the frame latency. We expect that, with the advancements in hardware accelerators for vision algorithms (e.g., DNN accelerators), the front-end will be more important, widening the applicability of SVSoC.

SVSoC in its current implementation does not target mission-critical systems that have tight accuracy requirements. SVSoC could be used in general vision-based systems such as AR and robotics navigation that are less accuracy sensitive. Critically, through the oracle data we show that the latency and accuracy of SVSoC would improve as the prediction accuracy improves. Thus, SVSoC can readily benefit from a higher quality frame predictor with the algorithmic innovations from the vision community.



(a) Front-end-dominate case. (b) Vision-stage-dominate case.

Fig. 7: Latency reduction varies with energy budget

.

6 Conclusion

We identify the sequential execution model as the root-cause of long frame latency in continuous vision systems. Our preliminary results show that speculatively execution the vision pipeline with proper architectural support could reduce the frame latency while consuming lower energy, enabling more agile vision-based systems.

References

[1] "Open-Q 820 Development Kit," https://bit.ly/2JAjbNs.

[2] "Qualcomm Snapdragon 835 First to 10 nm," https://bit.ly/2OnPD6u.

[3] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. springer, 2009, vol. 56.

[4] A. Censi and D. Scaramuzza, "Low-latency event-based visual odometry," in *Proc. of ICRA*, 2014.

[5] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[6] 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.

[7] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv preprint arXiv:1605.08104*, 2016.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. of CVPR*, 2016.

[9] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator," *arXiv preprint arXiv:1811.02883*, 2018.

[10] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[11] Y. Zhu, A. Samajdar, M. Mattina, and P. Whatmough, "Euphrates: Algorithm-soc co-design for low-power mobile continuous vision," in *Proc. of ISCA*, 2018.