Platform-Aware Resource Allocation for Inference on Multi-Accelerator Edge Devices

Ramakrishnan Sivakumar
Intel Corporation
ramakrishnan.sivakumar@intel.com

Saurabh Tangri
Intel Corporation
saurabh.tangri@intel.com

Satyam Srivastava
Intel Corporation
satyam.srivastava@intel.com

Abstract

With the fast pace of innovation in deep learning (DL) and machine learning (ML) algorithms, large datasets, and capable hardware, real-world applications are adopting artificial intelligence (AI) even on personal computing devices. These devices tend to run many applications in parallel and try to offload compute to the cloud. Lately, a lot of effort has been made to do more AI compute locally using available device hardware to provide better privacy, improved reliability, and lower costs. To enable optimal AI to compute on the local device, hardware vendors have resorted to integrating AI accelerators on-chip. With this adoption, the traditional system-on-chip now houses more than one compute entities for AI acceleration. In this paper, we highlight how such a multi-accelerator personal computing system is underutilized and results in sub-optimal performance without the right scheduling interfaces. We propose a scheduling interface that understands the current system state, user preferences, and specific workload characteristics to guide applications and AI frameworks to effectively utilize the underlying hardware resulting in improved user experience. We propose a holistic measure of performance rather than measuring peak steady-state throughput as the performance indicator. Initial experiments show that such an approach can provide up to 40% improvement in system responsiveness in simulated usage of real-world applications.

1 Introduction

Most prominent ML applications rely on the cloud for compute scaling and ease of deployment but with an increase in solution costs, security implications, and reliability issues the industry has been moving towards deployment on the edge. By deploying such services directly to edge devices ML applications can mitigate these while balancing local compute and cloud offload [12] [10]. Many cloud-based ML tools make it easy to experiment, train, deploy, and service DL applications. However, porting the model and the inference runtime to edge poses a significant challenge. The inference runtime tasked to evaluate models is highly optimized to saturate all available resources when running on the cloud. These runtimes implement custom thread pools for server-class CPUs and implement packed layout schemas to improve data locality for bigger caches when performing matrix math [6]. There is also complex graph compiler-based techniques that have produced impressive results for both cloud and edge workloads.

While the above optimizations lead to high throughput, these can be detrimental to the overall behavior of a user or edge device where many applications contend for limited resources. When optimizing DL

6th Workshop on Energy Efficient Machine Learning and Cognitive Computing (EMC2 2020), Online (San Jose, California).

libraries for edge devices it is very important to optimize for overall system performance and energy efficiency. Inference runtimes also need to track increases in startup latencies that can impact the user experience. In this paper, we propose broadening the evaluation of ML systems to deliver the greatest ML throughput with minimal disruption to the user experience. We also propose a simple algorithm to determine system resource allocation for inference based on available compute and memory, ML model characteristics, runtime, and user preferences.

2 Existing Methods for ML Performance Evaluation

While the field of machine learning has matured considerably and been widely deployed, the methods used to quantify efficiency and performance are largely rooted in research. This is evident from most articles that describe the "performance" of an ML system solely in terms of accuracy [3]. In recent years, there have been efforts (academic, industry, and collaborative) to represent both the effectiveness and efficiency of an ML system in real-world usage. DeepBench provides benchmarks to assess training and inference on various frameworks [7]. It evaluates computational metrics for fundamental ML operations (such as convolution) on various hardware platforms. CEA N2D2 is an open-source benchmarking framework that simulates the performance of DL models on various hardware configurations [5]. It enables designers to explore and generate DL models and compares different hardware based on DL model accuracy, processing time, hardware cost, and energy consumption. Stanford's DAWNBench provides a reference set of common DL workloads for quantifying training time, training cost, inference latency, and inference cost across different optimization strategies, model architectures, software frameworks, clouds, and hardware [1]. The MLPerf group provides a broad set of benchmarks for training and inference that evaluates hardware based on accuracy, speed, and cost of operation [8]. The AdaBench proposal aims to build a standard approach to benchmarking of complete ML pipelines across different applications, platforms, and problem sizes [9].

In nearly all the methods for evaluating performance and efficiency, the goal is to maximize the speed of training or inference (throughput). This approach works well for large-scale cloud systems where the fastest time to complete a task is most important. Most methods fall short of recognizing the vastly different usage patterns of a personal computation device (such as a PC or a mobile phone). For example, if we compare two software frameworks (A and B) to complete the same inference task on a PC the more aggressive framework A finishes faster, resulting in a higher score on the MLPerf inference benchmark. However, upon deeper analysis of system resources, we observed that framework A spawns a large number of idle spinning threads which starves many user activities. The more modestly configured framework B spawns fewer, passive threads which do not starve user applications and still produce adequate throughput. Therefore, there is a need for defining metrics that evaluate ML performance on a user-facing device and to use those metrics for optimal allocation of resources.

3 IRIS: Improved Responsiveness for Intelligent Systems

The performance of a user device should be defined by how responsive the platform is. This is based on how the resources available are shared by different applications running at any time. The current state of a given platform is dependent on *platform configuration*, *application configuration* and *Workload configuration*. Platform configuration refers to the invariant components of a system, both static (number of cores, available accelerators, total physical memory size and bandwidth, power supply or battery capacity, available instruction set and frameworks) and dynamic (current core and memory utilization, the existence of high priority workloads, current power settings). Application configuration is influenced by user and developer preferences. User preferences are factors that vary based on how the user intends to use a given application and how that impacts the user experience. For example, a given application when used in the foreground of the system impacts the quality of experience directly while the same application active in the background does not impact the user experience. The intention of the user decides the relative priority of the application among other active applications. Developer preferences are factors that the application developer controls such as the choice of computing device, runtime, and ML model used to perform a specific task. Workload configuration depends on the complexity of the workload that influences the amount of system

resources required to complete the workload in a reasonable time. Next, we propose an algorithm to recommend the optimal workload configuration focused on enhancing user experience.

We consider a problem where an inference needs to be executed on a platform with a traditional CPU, GPU, and a low-power custom ASIC designed for AI acceleration. The goal is to determine the optimal target of execution and allocation of system resources that maximizes the quality of user experience. Our method determines resources required for an ML application using –

- Selections made by a user and the ML runtime indicate their preference for speed or efficiency. A user may indicate high performance or low power mode of operation. And preferences on Latency sensitivity and FPS specification between Real-time, Foreground, or Background.
- Pre-calculated heuristics computed over a large set of representative topologies provide ideal resources for an unconstrained execution. These also include theoretical or roof-line analysis to determine the compute vs memory boundedness of topology on the given accelerator.
- Platform configuration and current utilization determine rough upper bounds on the allocable resources. While it may be possible to evict certain processes in favor of high priority inference, it is not covered under the current method.

Let X, Y, and Z represent the platform, application, and workload factors described above. These are defined as follows –
$X = \{x_{cu}, x_{tm}, x_{ut}, x_{um}\}$
$Y = \{y_{pp}, y_{fs}, y_{ls}\}$
$Z = \{z_{bo}\}$
The output of the algorithm is $S = \{s_{ip}, s_{cu}, s_{cp}, s_{mf}\}$. Here:

Table 1: Definition of inputs to IRIS

Name	Description	Values
$[IP]x_{cu}$	Total number of compute units	$x_{cu} \geq 1$ (Integer)
$[IP]x_{tm}$	Available memory in gigabytes	$x_{tm} \geq 0$ (Real)
$[IP]x_{ut}$	Current %utilization of the accelerator	$0.0 \leq x_{ut} \leq 1.0$ (Real)
$[IP]x_{um}$	Current memory utilization	$0.0 \leq x_{um} \leq 1.0$ (Real)
y_{pp}	User selected power plan (low/high)	$y_{pp} \in \{0, 1\}$ (Boolean)
y_{fs}	FPS specification	$y_{fs} \in \{0, 1, 2\}$ (Integer)
y_{ls}	Latency specification	$y_{ls} \in \{0, 1\}$ (Boolean)
z_{bo}	Model-specific compute boundedness	$0.0 \leq z_{bo} \leq 1.0$ (Real)
$[IP]s_{ip}$	Score per accelerator	$0.0 \leq s_{ip} \leq 1.0$ (Real)
$[IP]s_{cu}$	Suggested compute unit allocation	$0.0 \leq s_{cu} \leq 1.0$ (Real)
$[IP]s_{cp}$	Suggested context priority	$s_{cp} \in \{0, 1\}$ (Boolean)
$[IP]s_{mf}$	Suggested memory footprint	$0.0 \leq s_{mf} \leq 1.0$ (Real)

Based on the above figure, the decision algorithm is defined as follows.

$s_{ip} = f(X, Y, Z)$
$s_{ip} = Wc.f(x_{cu}, x_{ut}) + Wm.f(x_{tm}, x_{um}) + Wu.y_{pp}$
Where, W = Weight associated with each metric for a given IP

Table 2: Definition of Weights

Name	Description
$Wc = z_{bo}$	Compute weight determined by the compute boundedness of the model.
$Wm = 1 - z_{bo}$	Memory weight is defined as (1 - compute boundedness).
$Wu = f(y_{fs}, y_{ls})$	User setting weight defined as a function of FPS and Latency specification.

$s_{cu} = z_{bo}.f(x_{tm}, x_{um}, y_{fs}, y_{ls})$
$s_{cp} = f(y_{fs}, y_{ls})$
$s_{mf} = (1 - z_{bo}).f(x_{cu}, x_{ut}, y_{fs}, y_{ls})$

The compute or memory boundedness of the model plays a major role in influencing the score for the IP with respect to the available compute or memory resources for that IP.
Note that if an app developer chooses to *not* utilize IRIS, their requests will override the suggestions. This ensures that our solution does not interfere with the ability of a knowledgeable developer to create a good experience for their user. Second, when inference sessions run longer than a certain time duration, the method is re-run to compute resource allocations based on the new state of the system.

3.1 Heuristic Analysis

The decision on the amount of compute and memory resources provided to an inference session depends on the number of ways the model can be split for concurrent execution. Explicit analysis of dependencies across model parameters provides us with the optimum amount of parallelism the model can benefit from [4]. At a high level, model parameters are the attributes of a given model such as the number of MACs (multiply accumulates), subgraph count, and op level complexity. Using the optimal number of threads prevents wastage of system resources handling the additional threads as well as reduces the inter-core data movement on multi-core platforms in turn reducing communication traffic that may lead to bandwidth saturation [11].

In terms of memory usage, the best-case scenario is to fit the complete model in memory for the total execution time. This guarantees minimal access times by preventing page faults. Heuristics analysis provides an estimate of the maximum required memory by reading the input dimensions, the architecture of the models, and the type of data being used. IRIS makes use of the above-reported values to scale the number of threads and memory provided for the inference with respect to maximum threads and memory footprint of the computation. We perform the heuristic analysis only if the maximum FPS or best possible latencies are requested. By default, thresholds are set to 25% of total compute units available and 25% of total system memory.

4 Experimental Results

We define user experience as the overall experience of using a product or application in terms of responsiveness and ease of use.

4.1 Measuring Impact on System Responsiveness - Gaming + Streaming

As one of the leading usage scenarios of edge devices, we focus on Gaming + Streaming as a use case to demonstrate the functionality of IRIS. In this experiment, we simulate the user playing a graphics-intensive game (Dues Ex: Mankind Divided) in the foreground and also streaming the gamer playing the game which is a common usage scenario on platforms such as Twitch. This live streaming of the gamer also includes the background blur feature for privacy reasons. Background blur involves segmenting the user from the background and applying a blur filter to the background. This is performed on each frame received from the webcam facing the gamer using the DeepLabV3 model. This experiment is performed on a 4-core Intel KabyLake platform with Intel(R) Iris Pro Graphics 580. Intel OpenVino is used as the framework on which the Background segmentation application is built on.

Figure 1 plots the sorted amount of time taken to render each frame of the game when a) The game is played without any background segmentation which acts as the baseline and b) The game is played along with the background segmentation executing on the GPU c) The game is played along with the background segmentation executing on the GPU. We would like to focus on the farther end of the graph that points to highlights the frames with the worst render times in all the 3 cases. In the baseline case (a) the 99th percentile render times are in the ranges of 60-70ms per frame, whereas when the segmentation happens on the GPU (c) the 99th percentile render times regress to 170 - 180ms per frame. This is a 2.14x regression in the gaming performance compared to the baseline case. This is because the graphics-intensive game is competing for GPU resources with the background segmentation process thus losing performance. More often than not application developers tend to choose the GPUs for inference due to the general notion that GPUs provide higher parallelism and throughput compared to CPUs. But this does not take into account the other applications running on the platform and how choosing the GPU can impact the end-to-end user experience.

4.2 Responsiveness with Recommendations from IRIS - Gaming + Streaming

Next, we run the same set of concurrent applications on the same client system but with resource allocation honoring the guidance provided by IRIS. The following are the inputs to IRIS for selecting the IP to execute the background segmentation. Test System comprises 8 CPU threads and 4GB of memory with both the CPU and memory being used up to 60% by the game at this time. ($[CPU]X = \{8, 4, 0.60, 0.60\}$). The Intel(R) Iris Pro Graphics 580 has 48 Execution units with access to 2GB of memory and is utilized up to 90% at this time. ($[GPU]X = \{48, 2, 0.90, 0.90\}$). The user and runtime configuration includes power plan set to high performance, FPS specification set to 1 (medium) and Latency specification set to 0 ($Y = \{1, 1, 0\}$), and the compute boundedness of DeepLabV3 model set to 0.8 indicating high compute boundedness due to the convolution heave nature of the model ($Z = \{0.8\}$). Based on the above inputs IRIS recommends an allocation.

[CPU]S = {1.0, 0.25, 1, 0.25}
[GPU]S = {0.0, 0.25, 0, 0.25}

IRIS strongly recommends the use of CPU in this particular use case with a CPU score of 1.0 and a GPU score of 0.0. Within the CPU IRIS recommends allocating 25% of the available memory and 25% of the available compute units (CPU cores) for this inference. Based on the above recommendations provided by IRIS, the background segmentation is now executed on the CPU to prevent the contention of GPU resources that are being utilized by the game. Figure 1 outlines the results of gaming performance when the background segmentation is executed on the CPU compared to when the background segmentation is executed on the GPU. From the 99th percentile render times we observe a significant reduction in gaming performance regression from 2.14x to 1.52x.

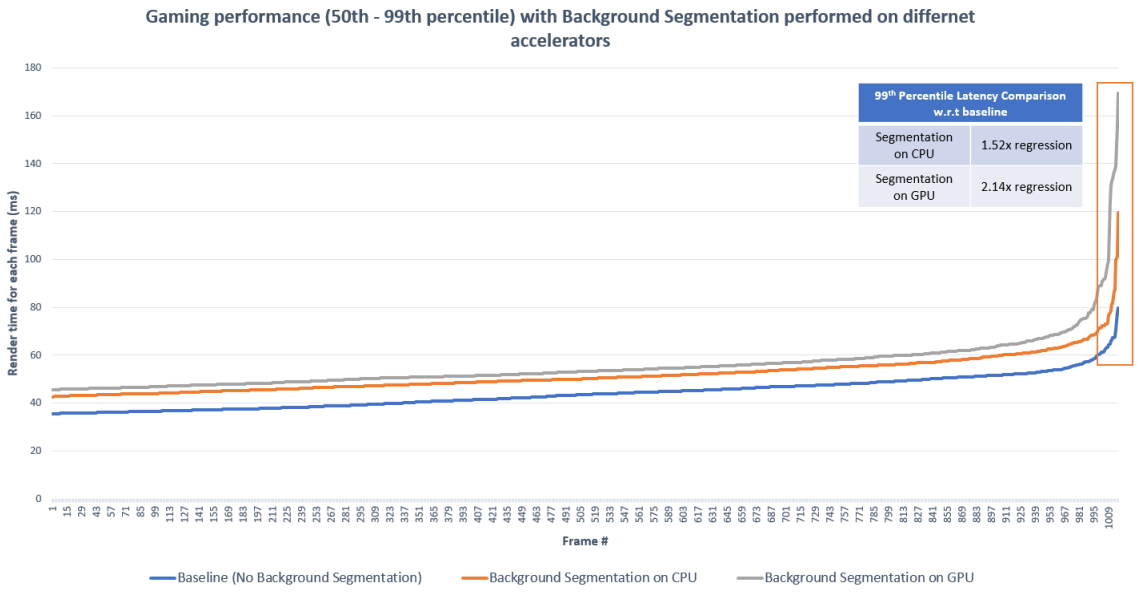

Figure 1: Gaming perfromance for Gaming + Background Segmentation on CPU and GPU

It can be seen that a more thoughtful allocation of resources for inference leads to improvements in both gaming and background segmentation performance thus improving the overall user experience. These are achieved without a significant drop (under 15%) in the inference throughput. In some cases, the improvement in responsiveness can be large especially when workload options allow for more stingy configurations. This simply cannot be done with normal runtimes which are designed for high throughput irrespective of system state.

4.3 Measuring Impact on System Responsiveness - Productivity + Inference

As another leading usage scenario of edge devices, we focus on Productivity and break it down into tasks to help quantify the user experience. It comprises word processing, presentation, spreadsheets, and email clients. Each of the sub scenarios consists of a sequence of actions defining a user's interaction which are automated through scripts. We build instrumentation in the system to measure the time taken to complete a particular operation. Some of the tasks that we track are: application launch time, opening drop-down menus, converting files to different formats, opening files saved on the disk, plotting graphs, media processing. The industry is moving towards integrating these basic productivity tasks with intelligent features based on ML. To simulate concurrent productivity and ML we added a continuous background inference (image classification with SqueezeNet) to foreground productivity tasks using Windows ML [2] at default settings. The graphs in Figure 2 show anywhere between 70% to 120% increase in time taken to launch the application and between 20% to 115% longer time to complete the previously defined tasks. Runtimes tuned to achieve the highest throughputs with their default settings will cause adverse effects on the user experience of the device.

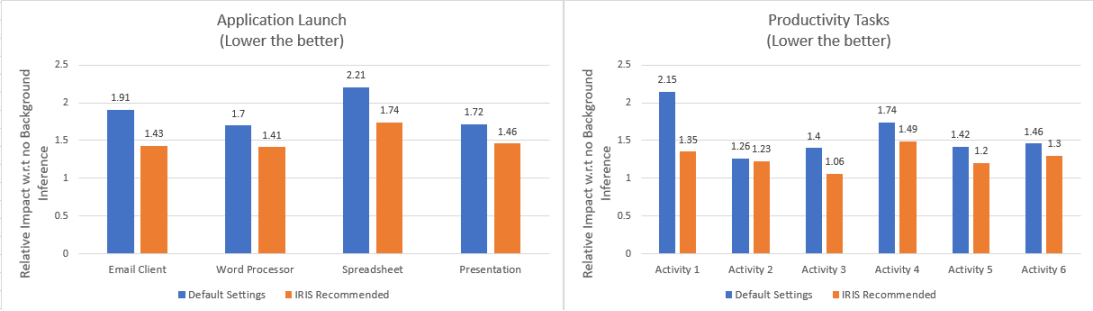

Figure 2: Relative performance comparison of productivity tasks with concurrent background ML processing

4.4 Responsiveness with Recommendations from IRIS - Productivity + Inference

Next, we run the same set of concurrent applications on the same client system where the GPU is unavailable for use. We test how IRIS handles resource allocation when only the CPU is available for use. Client System used is a laptop comprising 8 CPU threads and 4GB of memory with both the CPU and memory being used up to 25% by other applications at this time ($[CPU]X = \{8, 4, 0.25, 0.25\}$). The user and runtime configuration includes power plan set to high performance, FPS specification set to 0 indicating no requirement of high throughput, and Latency specification set to 1 indicating that this inference for the productivity tasks needs to be completed with low latency($Y = \{1, 0, 1\}$), and the compute boundedness of SqueezeNet model set to 0.5 indicating moderate compute boundedness ($Z = \{0.5\}$). The heuristics analysis recommends ideal parallelism of 4 threads and a memory footprint of 2GB for optimal inference performance. Based on the above inputs IRIS recommends an allocation $[CPU]S = \{1.0, 0.5, 1, 0.5\}$.

5 Conclusions and Future Work

In this paper, we present a detailed analysis of quantifying machine learning inference performance on client and edge devices. We first survey existing methods of performance evaluation and observe that most such metrics are centered around peak steady-state throughput. We demonstrate experimentally that such techniques lead to adverse effects on system responsiveness/ user experience which is critical in user devices. Finally, we propose a method for inference system resource allocation that leads to a large improvement in system responsiveness without a significant drop in inference throughput.

In the future, this method can be extended to take all the metrics into considerations for superior resource allocations. In addition to the static parameters described above, it is also possible to include a feedback mechanism that would adjust the decision thresholds of the algorithm. This mechanism

could consume data available through telemetry and may use reinforcement learning methods to arrive at the best parameters during use.

References

[1] Cody A Coleman et al. Dawnbench: An end-to-end deep learning benchmark and competition. NIPS ML Systems Workshop, 2017.

[2] Microsoft Corp. Windows ML. `https://github.com/Microsoft/Windows-Machine-Learning`, accessed 2019.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[4] Seunghak Lee, Jin Kyu Kim, Xun Zheng, Qirong Ho, Garth A Gibson, and Eric P Xing. On model parallelization and scheduling strategies for distributed machine learning, 2014.

[5] CEA LIST. Neural network design & deployment. `https://github.com/CEA-LIST/N2D2`, accessed 2019.

[6] Yizhi Liu, Yao Wang, Ruofei Yu, Mu Li, Vin Sharma, and Yida Wang. Optimizing cnn model inference on cpus. arXiv:1809.02697v3, 2019.

[7] Sharan Narang. Deepbench. `https://svail.github.io/DeepBench/`, accessed 2019.

[8] David Patterson et al. Mlperf: A benchmark suite for machine learning from academic-industry cooperative, 2018.

[9] Tilmann Rabl et al. Adabench - towards and industry standard benchmark for advanced analytics. In *TPC Technology Conference on Performance Evaluation & Benchmarking*, 2019.

[10] Zhi Zhou et al. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107, 2019.

[11] Kaiwei Zou, Ying Wang, Huawei Li, and Xiaowei Li. Learn-to-scale: Parallelizing deep learning inference on chip multiprocessor architecture, 2019.

[12] Zhou Zou et al. Edge and fog computing enabled ai for iot -an overview, 2019.