Method

In this power-performance characterization report, we evaluated the power and performance of five state-of-the-art image classification Convolutional Neural Networks (CNNs) on the Khadas VIM 3 board. The CNNs used in this study were AlexNet, GoogleNet, MobileNet, ResNet50, and SqueezeNet. These CNNs were executed using the ARM-CL framework on an ARM-based Heterogeneous Multi-Processor Systems on Chip (HMPSoC) known as the Amlogic A311D HMPSoC, which is integrated in the Khadas Vim 3 embedded platform.

The HMPSoC used in this study contains a hexa-core asymmetric ARM big-little multicore CPU with two CPU clusters, Big and Little. The quad-core big CPU cluster contains four high-power, high-performance A73 cores, while the dual-core Little CPU cluster contains two low-power, low-performance A53 cores. The HMPSoC also contains a dual-core Mali G52 MP4 GPU. The maximum frequency for the big CPU cluster, Little CPU cluster, and GPU is 1.8 GHz, 2.2 GHz, and 0.8 GHz, respectively. A 4 GB LPDDR4 main memory supports the HMPSoC. In software, the platform was running Android v9.0 with kernel v4.9. We ran ARM-CL v21.02 on top of it.

To perform the power-performance characterization, we used a modified research version of the ARM-CL library called Pipe-All (Aghapour, \& Torab, 2022). This version of the library allows for simultaneous inference on both CPU and GPU cores using a software pipeline. The parameterized code for Pipe-All allows the user to redistribute the workload between the big CPU cluster, the Little CPU cluster, and the GPU. The Governor used in this work used this workload distribution feature from Pipe-All as a power management knob. In addition to the workload distribution feature, we also used Dynamic Voltage and Frequency Scaling (DVFS) as a power management knob. DVFS technology changes a core’s voltage and frequency at run-time, allowing Governors to tailor the performance of a core to match the requirements and save power. To physically set up the board, we followed the instructions provided in the guide. This included using a laptop running Ubuntu 20.04 and a USB-A port to connect to the Khadas VIM3 board, as well as other required peripheral equipment.

We collected power and performance data for each CNN using the governor and power management knobs described above, by recording the minimum, maximum, and median power consumption (W) and the average frame latency during the inference interval. The numbers of frames used for the Little CPU, Big CPU, and GPU are 80, 100, and 100 frames consecutively. Every CNN was run on a combination of pipelines in which each cluster was configured as the first target in the pipeline. For the Little and Big CPUs, the maximum frequency level was also changed incrementally based on the permitted options of frequencies for each of the CPU clusters, to record the power-performance data to analyze the complete relationship between power consumption and performance level. Furthermore, These data were then used to create charts and graphs to visually represent the power and performance characteristics of the CNNs on the Khadas VIM 3 board.

In order to gain a deeper understanding of the impact of ordering on performance and energy consumption, we conducted a series of tests regarding the pipeline’s structure, which can be separated into two kinds of experiments.

First, we changed the placement of the three clusters in the pipeline while maintaining the same workload. To ensure consistency, we fixed the frequency at 1704000 Hz for the Big A73 cores and 1200000 Hz for the Little A53 cores. We then partitioned the CNN parts into even blocks, and systematically evaluated each core alone and in various combinations of two and three cores. This allowed us to gain a comprehensive insight into the optimal partitioning of the CNN parts and the associated performance and energy consumption for the different CNNs we tested.

Then, we decided to observe the impact of particular divisions of the parts on power-performance relationship by abstaining from the previous ‘equal partitioning’ method. Due to large possibilities of dividing the parts, it was logical to divide the number of parts by half, allocate one half to a position, and divide the other half to the remaining two locations in a similar manner. For the case of MobileNet for instance, we started with 28-0-0, then six permutations of 14-10-4, and divided 14 by 2 to create six permutations of 7-14-7. This allowed us to minimize the negative impact of the lack of data acquisition on our evaluation of the effect of partitioning by generating eleven reasonably spreaded cases in the design space.

In this report, we provided generalized insights on how the power and performance move with different power-efficiency knobs in the board. The results of this study can be used to inform the design of governors and power management strategies for embedded systems operating in power-constrained environments.

Results

Following the experiment, the data collected were illustrated in several plots in order to make some observations. Firstly, the general form of the relationship between the power usage of CNN and latency is shown in Figure 1.

-------------------------------

Figure 1, 2

-------------------------------

In Figure 1, the graph shows that as performance increases the more it increases the more energy it cost to increase the graph form. this form with only the time was also found Resnet50 and SqueezeNet. This is the most basic graph. the only real difference is the latency scale where they work on where on Restnet50 200-800ms and SqueezeNet is 80-225ms.

In Figure 2, we note the same form as in Figure 2, only now with fewer intervals and the numbers being lower. Nevertheless, for the rest of the plots, we can see it in the last graph how these normally look next to each other. the other graphs are very similar to the all the other graphs in this category.

-------------------------------

Figure 3, 4

-------------------------------

As evident from the figure, there is a big change in performance, even on low performance we can see that it has a bit in terms of latency. This means we better be running on low on the big cpu then high on the low cpu. this is true for almost any. except that of AlexNet witch is the one we haven't covered in this piece yet. this cause alexnet does something a bit weird.

In Figure 4, there is a fluctuation at the centre of the normally quite good looking slope. this is also findable in the GoogleNet one on the same frequency. we also find that the latency plot has alot of overlap. this did not happen on the other algorithms.

The last thing we tested was the GPU performance but this was a very not super interesting graph on it own so we decide to look at max min and GPU how many fps/ w the generate because FPS = 1/throughput we can say this it efficient they are.

-------------------------------

Figure 3, 4

-------------------------------

Figure 5 show how efficient the GPU is if you just want the frames per second for wats used. other stuff is the big cpu looks a lot more power efficient if you need the speed it overs.

The first thing that we noticed doing with the thing that was noticed in the test that we see something that I would call stall. it is a phenomenon that happens when a faster core is after a way slower core. When this happens what you see happen is long wait times at the faster core with we didn't record why we do this in text and a temporary loss of power of the W we measured as the faster core went to wait on it's next batch. what this means is that these stalls give a good return on average watt but take a good hit in performance in latency and fps. Another way is the other way around when the faster core was before the slower core meant that the faster core often was quickly done and only the slower core had to do work this resulted in the same sort of power drop after the faster core was done.

When we looked at our second set of experiments regarding the order of the processors, we obtained these observations. First, we considered the changes in position of the idle or of the working processors. When all parts are allocated in one of the processors, the total time remained constant regardless of the order of the remaining processors. (Please give a quantitative example from the data above). However, when the orders of the working processors in the two-processor and three-processor pipelines are changed, various impacts on inference time can be seen. First, whenever the Little core is placed before either the other two processors, an apparent changes in total inference time were observed. On the other hand, when the GPU and Big CPU were swapped around, the difference in the change in performance is generally minimal, although there were cases of notable differences.

As a result, one notable observation from the tests was the occurrence of a phenomenon referred to as "Process Bottleneck" or "Congestion", where a less powerful process becomes a bottleneck as it is placed before a more powerful one in the pipeline. This results in long wait times at the more powerful core, resulting in a temporary loss of power and a decrease in performance and latency. This effect was also observed when the more powerful core was before the less powerful core, with the more powerful core completing its tasks quickly and the less powerful core being left to do the remaining work, resulting in a similar power drop. Essentially, the cycle time increases as the workloads resulted from the partitioning are unequally allocated between stages, resulting in process bottleneck. This phenomenon can be observed from Figure 6 and 7 of the total inference times of each processor when AlexNet was used.

Compared to when the Big CPU had to work alone by taking responsible for all the parts, when the total workload were divided evenly with the GPU, it was able to reach the same performance with the GPU. For instance, in Figures 6 and 7, when the number of parts are equal in both the Big CPU and GPU, the Big CPU most often completed in the same amount of inference time as the GPU (e.g., all pairs of bars in 4-4-0 and 3-3-2, for instance, GBL vs. BGL). However, this positive performance dissipated in the case of MobileNet. Regardless of having the same number of parts (e.g, LGB vs. LBG of MobileNet), the total inference times between the 2 cases were clearly different. This showed that the difference in the number of operations in certain parts of a CNN has impact on the performance and process time of the pipeline, and led us to further investigate into the partitioning of the CNNs’ parts to evaluate the number of operations in areas of the CNN.

Another point to note is from Figure 6 and Figure 7,

Example of GoogleNet. Because