Universiteit van Amsterdam

Assignment 2

Power-Performance Characterisation Report

20 januari 2026

Tutor:
Hang Xu

Practicumgroep:
Group 11

Docent:
Anuj Pathania

Cursus:
Embedded Systems

Vakcode:
5063EMSY6Y

Studenten:
Adam Rakun
16339908

Matteo D'Urso
16349210

1 Introduction [0.5 Points]

In this report, we perform a systematic power–performance characterization of AlexNet inference on the Khadas VIM3 platform, which features an ARM big.Little CPU architecture and a Mali GPU. The inference is executed using the Pipe-All framework, which enables mapping of neural network workloads onto the available processing elements.

Our goal is to quantify how different power-efficiency knobs (namely CPU frequency scaling, pipeline partitioning, and component ordering) affect the three primary perfomance metrics: throughput (FPS) , latency (ms), and overall system power consumption (Watts).

More precisely, we will measure these three values while scaling CPU's frequencies and changing pipeline's partitions, so that we get the following graphs and answer the hypotheses below:
1. Power-Performance graphs for *AlexNet* on *Little* CPU cluster at different frequencies.
2. Power-Performance graphs for *AlexNet* on *big* CPU cluster at different frequencies.
3. Power-Performance graphs for *AlexNet* on *GPU* at different *big/Little* CPU frequencies.
4. Power-Performance graphs for *AlexNet* for different partition points, at fixed frequencies and fixed component order.
5. Power-Performance graphs for *AlexNet* different component orders, at fixed frequencies and fixed partition points.

Before conducting the experiments, we formulate the following hypotheses for the five points above:

- **H1:** Increasing the operating frequency of the Little CPU cluster will result in a gradual increase in power consumption while providing moderate performance improvements, reflecting the energy-efficient (low power, low efficiency) design of the Little cores.
- **H2:** Increasing the operating frequency of the Big CPU cluster will lead to higher power consumption compared to the little cluster, especially at high frequencies (because of its high power - high performance design). We expect that latency decreases and throughput

increases with an increasing frequency. At high performance points, we expect diminishing returns of performance with respect to frequency.

- **H3:** When AlexNet is executed entirely on the GPU, variations in big and little CPU frequencies will still have an effect on power consumption and performance because of the overhead (memory, synchronization, preprocessing...) carried out by the CPUs as the GPU is not a standalone processor. Nevertheless, we expect the effect to be almost negligible.
- **H4:** Different pipeline partition points will expose distinct power–performance trade-offs which rely on AlexNet's design. Early convolutional layers of AlexNet are computationally intensive and well suited for GPU execution, while later layers are lighter and more appropriate for CPU execution. Assigning too few layers to the GPU is expected to underutilize its parallelism. On the other hand, assigning too many layers may underutilize the CPUs' performances.
  We therefore hypothesize that an intermediate partitioning, where the GPU processes early layers, the Big CPU processes intermediate layers, and the Little CPU processes final layers, will achieve the best power–performance trade-off.
- **H5:** Changing component orders with fixed pipeline partition points and CPU frequencies is expected to significantly affect both performance and power. Component orders that assign computationally heavier stages to faster processing elements (such as the GPU or Big CPU) earlier in the pipeline are expected to achieve higher throughput and lower latency, as this reduces the likelihood of a slow stage becoming the bottleneck.

The experiments presented in this report are designed to test these hypotheses by isolating different individual system parameters (frequencies, pipeline order and partitions) while keeping all other variables constant. The resulting power–performance graphs enable a good starting insight of trends, bottlenecks, and inefficiencies, which are essential for the following design of our effective runtime Governor that aims to meet performance targets while minimizing power consumption.

2 Experimental Setup [0.25 Points]

Hardware Platform

All experiments are conducted on the Khadas VIM3 development board, which is based on the Amlogic A311D heterogeneous multiprocessor system-on-chip (HMPSoC). The A311D integrates a big.Little CPU architecture consisting of four high-performance ARM Cortex-A73 cores (Big CPU) and two power-efficient ARM Cortex-A53 cores (Little CPU). The platform also includes a Mali GPU that supports general-purpose computation via OpenCL.

The heterogeneous nature of the platform enables multiple execution modes, allowing tasks to be mapped to either the Big CPU cluster, the Little CPU cluster, or the GPU, each offering different power–performance characteristics. Dynamic Voltage and Frequency Scaling (DVFS) is supported for both CPU clusters, enabling runtime adjustment of operating frequencies as a power-efficiency control knob. GPU frequency scaling, however, is not exposed on this platform.

The board is powered via an external power supply, and total system power consumption is measured at the input using an external USB power meter. This measurement includes the power consumed by all on-board components, such as the CPU clusters, GPU, memory subsystem, and cooling fan. To ensure consistency across experiments, the fan configuration is fixed throughout all measurements.

Software Workload

The workload used in this study is AlexNet, a convolutional neural network widely used as a benchmark for image classification tasks. AlexNet consists of a sequence of convolutional, pooling, and fully connected layers, making it representative of real-world deep learning inference workloads with varying computational intensity across layers.

Performance is evaluated using two primary metrics: throughput and latency. Throughput is measured in frames per second (FPS) and indicates how many input images the system can process per second. Latency measures the average time required to process a single input frame from start to finish.

Execution Framework

Inference execution is performed using the Pipe-All framework, which is built on top of the ARM Compute Library. Pipe-All enables the partitioning of a neural network into multiple subgraphs that can be executed concurrently on different processing elements of a heterogeneous system. Each subgraph corresponds to a pipeline stage that is mapped to either the Big CPU cluster, the Little CPU cluster, or the GPU.

The framework can divide the neural network into different pipeline stages, as well as configure a component order, which specifies the sequence in which processing elements appear in the pipeline. By adjusting these parameters, it is possible to shift computational load between processing elements and alter the pipeline bottleneck. AlexNet has 8 partition points by which it can be divided into pipeline stages. It is not possible to divide a single layer into different stages.

3 Methodology [0.25 Points]

For the experiments run to answer the hypotheses listed above, we used the following system setup.

- The fan of the Khadas VIM3 is set to constant speed, such that its energy consumption is constant.
- For each inference run, the number of frames is set to be at least 60 to get a reliable performance measurement. Numbers lower than 60 might lead to a false result as unexpected events influence execution.
- The power meter is read manually. To calculate the power consumption, we used the following formula: W = $(\frac{x}{1000})/(\frac{t}{3600})$, where x is the value in MWh ($1Wh = 3600J$) which we manually read from the power meter. For experiments where t or mWh were very small, we increased the number of frames to get more reliable measurements.
- The DVFS scaling of the system governor was set to manual and scripts were used to change the frequency during experiments

Regarding the choice of the fixed parameters in experiments we came to the following conclusions:

- **Experiments 1 and 2** have a straightforward setup. Both partition points were set to 8, to isolate the execution onto a single component. Then we ran inference for every frequency of the CPU while fixing the other CPU at its lowest frequency, in case it handled other (unrelated) tasks during execution.
- **Experiment 3** delegates the execution entirely to the GPU in a similar fashion as in experiments 1 and 2. However, as the cartesian product of the possible frequencies of the little CPU and big CPU is too big, we reduced the sample frequencies at which measurements are made. We only used frequencies with large increments in between them to get the bigger picture and cover the entire sample space, while keeping the sample size smaller. We believe that the overall pattern of the results can still be seen using fewer measurements and the findings remain reliable.
- **Experiment 4** is more complicated with respect to the choice of the setup of the experiment. As the space of possible variations is too large too even getting close to covering it exhaustively, we had to make some assumptions. Regarding the component order, we believe that the most reasonable order is GPU – big CPU – little CPU, as it resembles a *ground truth of efficiency.* The GPU covers the most intensive and parallelizable computations. The big CPU covers computations that are less intense and the little CPU covers the least intense computations. This claim will be further analyzed in the 5th experiment.

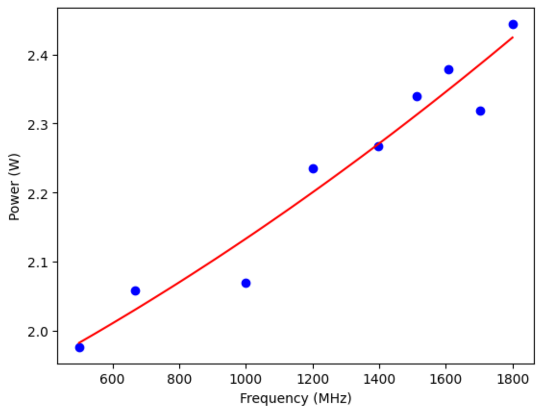
For a similar reason we used frequency values of 1200 MHz for the little CPU and 1800 MHz for the big CPU. Those values are in a moderate frequency range where the CPUs operate efficiently.
We then ran three *sub-experiments* to get the trade off between all component pairs. In sub-experiment (1) we fixed the size of the little CPU stage and evaluated trade-off between the big CPU and the GPU. We have done that by fixing the second partition point ($pp_2 = 6$) and iterating over through the possible positions of the first partition point ($0 \leq pp_1 \leq 6$)[1]. The (2)nd sub-experiment had a similar design as (1) with the first partition point $pp_1 = 2$ fixed and the second partition point $2 \leq pp_2 \leq 8$ variable. This fixed the the size of the GPU stage and evaluated trade-off between big CPU and little CPU. Similarly, for the last sub-experiment (3) we fixed the big CPU stage and evaluated trade-offs between little CPU and GPU. To do that we fixed an offset of 3 between the two partition points and iterated through the possible positions of the first partition point ($0 \leq pp_1 \leq 5$).
Furthermore, we are aware that keeping a pipeline stage to a constant size does not mean its impact is constant across all measurements and can therefore be disregarded. I.e. in (2) we fixed the GPU stage to size 2, and tried to isolate the big CPU and little CPU to analyze their trade-offs when varying the second partition point, however the GPU stage itself might still introduce behavior that is unrelated to the big CPU's and little CPU's trade-offs. We tried to account for that as much as possible and believe that results are still scientifically sound. Further experiments might be based solely on a 2-stage pipeline.

- **Experiment 5** is also complicated for analogous reasons of experiment 4. We chose the partition points to be $pp_1 = 2$, $pp_2 = 5$ as we wanted to achieve a balanced load distribution. As the first two layers are the largest, the first stage consists only of 2 layers. The remaining 6 layers were subdivided equally into 2 stages of 3 layers each. We chose the power frequencies of the big CPU and the little CPU to be 1800 MHz and 1200 MHz, for an analogous reason as in experiment 4. We ran all possible processor permutations as there were only 6.

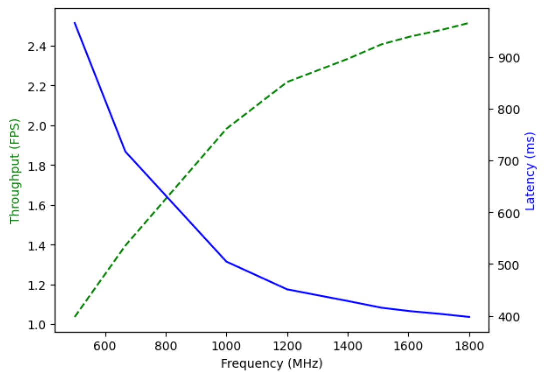
[1]Note: technically *Pipe-All* does not allow to fix a partition point on the 0th layer, however, that is semantically the same as fixing the partition point on the last layer and shifting the order. For instance $pp_1 = 0, pp_2 = 6$ with order G-B-L is the same as $pp_1 = 6, pp_2 = 8$ with order B-L-G as it associates the first 6 layers with the big CPU and the last 2 with the little CPU.

4 Results [0.75 Points]

1. Power-Performance graphs for AlexNet on Little CPU cluster at different frequencies.



(a) Frequency (MHz) vs Power (W) with the best fit line (red)

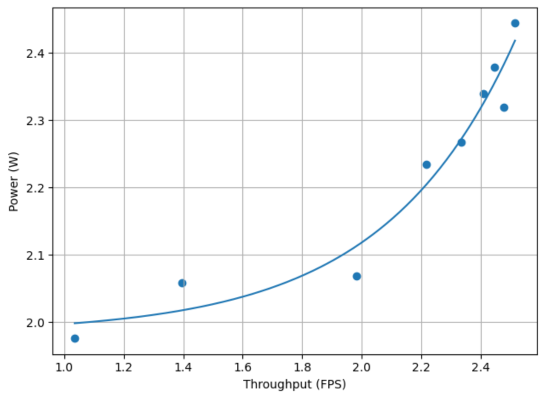


(b) Frequency (MHz) vs Latency (ms) (solid) and Throughput (FPS) (dashed)

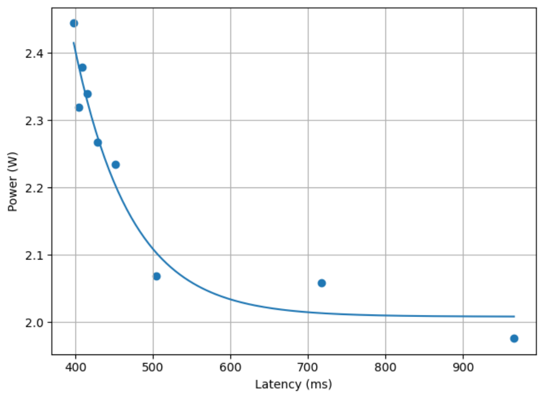
Figuur 1: Power-Performance graphs for the little CPU cluster

Figuur 1 shows that as the Little CPU frequency increases, the measured system power rises gradually and can be approximated by an almost linear trend.

Increasing Little CPU frequency improves throughput and reduces latency with diminishing returns at higher frequencies. Because power grows faster than FPS at high frequencies, we are interested in moderate (middle) frequencies for Little CPU.



(a) Throughput (FPS) vs Power (W)



(b) Latency (ms) vs Power (W)

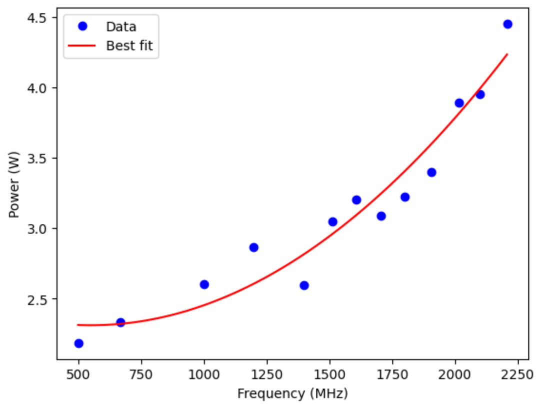
Figuur 2: Performance metrics vs Power on little CPU cluster. Graphs show an exponential growth

Figuur 2(a) shows a gradual trade-off: higher throughput requires higher power, with an gradually increasing slope. The slope however is less than on the big CPU. This indicates that at low power levels the Little cluster offers moderate performance gains with relatively low additional power.

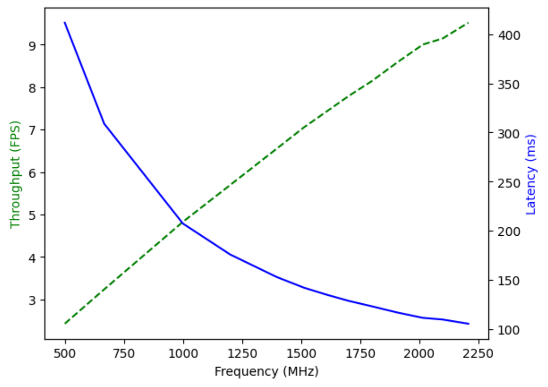
The power–latency relationship shown in Figuur 2(b) shows that reducing latency requires increased power, reflecting the typical energy–delay trade-off. At the lowest latencies, additional power produces smaller improvements, suggesting diminishing returns near the upper DVFS range.

Overall, Experiment 1 **supports Hypothesis 1**. Power increases gradually with Little CPU frequency while performance improves moderately. The relatively gentle power–performance slope confirms that the Little cluster is a suitable “energy-saving mode” for AlexNet when performance targets are met. We identify two practical implications for governor design: (i) the Little cluster provides predictable scaling and can be used to meet low-performance targets efficiently, and (ii) operating at the highest Little frequencies is often inefficient due to diminishing latency improvements per additional watt, so a governor should prefer mid-range Little frequencies.

2. Power-Performance graphs for AlexNet on Big CPU cluster at different frequencies.



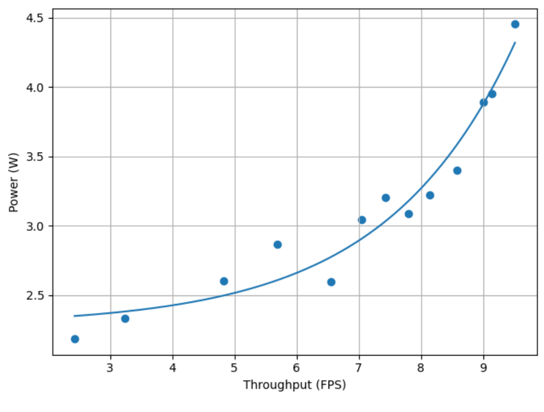
(a) Frequency (MHz) vs Power (W) with the best fit line (red)



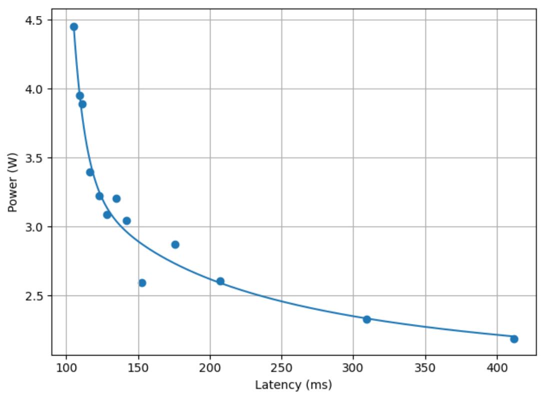
(b) Frequency (MHz) vs Latency (ms) (solid) and Througput (FPS) (dashed)

Figuur 3: Power-Performance graphs for the big CPU cluster

Figuur 3 shows that compared to the Little cluster, the Big cluster produces a much larger power swing across the frequency range, reflecting its high-performance microarchitecture and higher operating voltage requirements. Performance increases significantly as Big CPU frequency increases, demonstrating that AlexNet inference is strongly accelerated by the Big cluster. However, the curves exhibit diminishing returns.



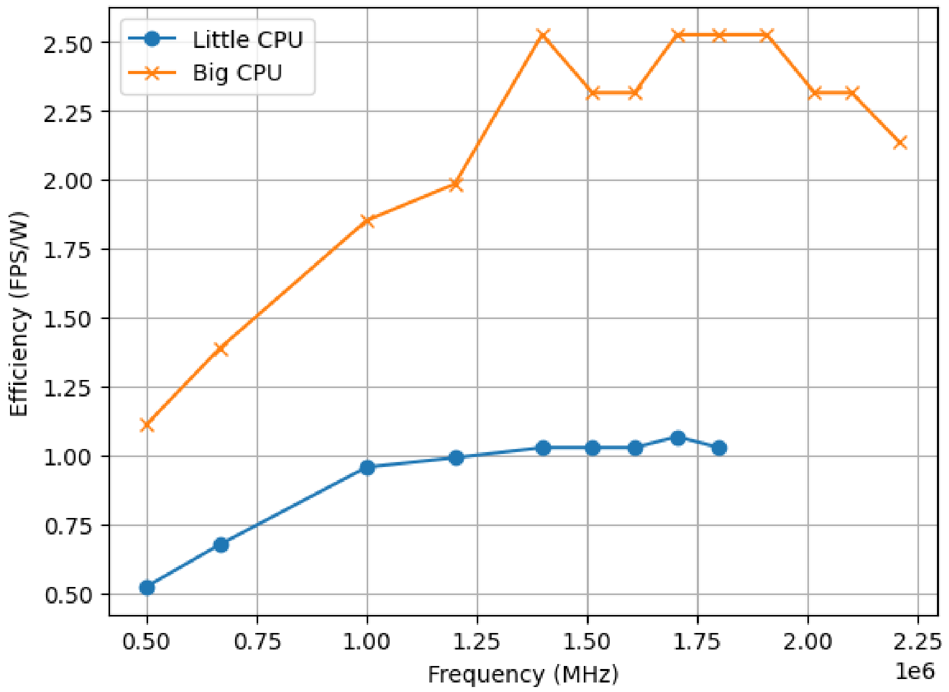
(a) Throughput (FPS) vs Power (W)



(b) Latency (ms) vs Power (W)

Figuur 4: Performance metrics vs Power on big CPU cluster. Graphs show an exponential growth

The power–throughput and power–latency curves are steeper than for the Little cluster as shown in Figuur 4, indicating that the Big cluster is effective for meeting high throughput requirements but becomes power-inefficient at the highest performance points.

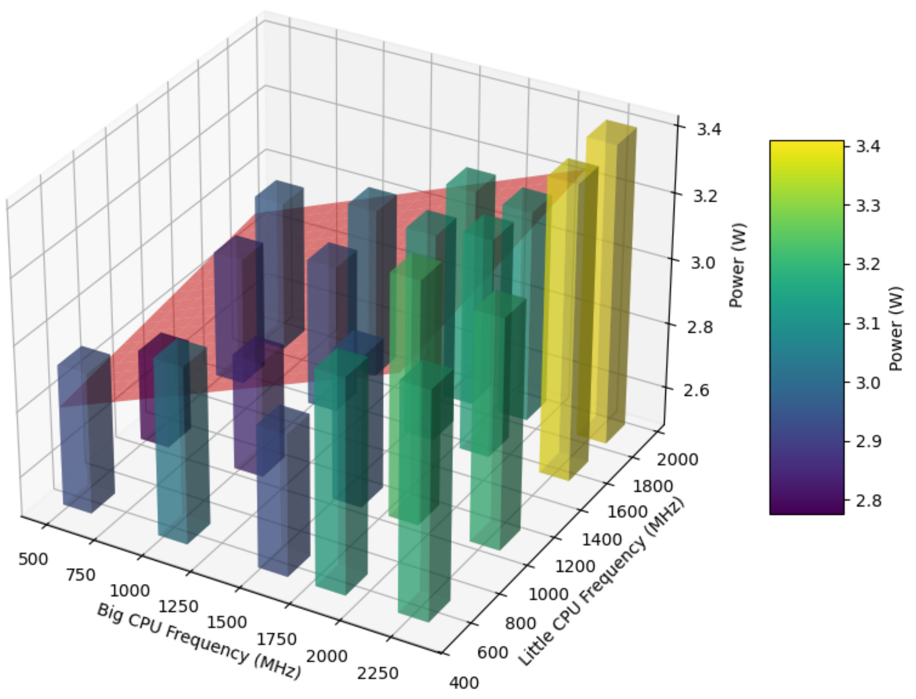


Figuur 5: Frequency vs Efficiency for little GPU cluster (dots) and big CPU cluster (crosses)

The efficiency comparison shown in Figuur 5 for Big and Little CPUs reveals that the Big CPU achieves substantially higher FPS per Watt than the Little CPU, meaning it delivers more useful work for each watt consumed. Big-core efficiency rises strongly from low frequencies and peaks around the mid–high frequency range, after which it declines again. This is consistent with Power vs Throughput graph above where power disproportionately increases at high FPS levels. In contrast, the Little CPU improves efficiency with frequency but saturates early at a much lower level, indicating that it is best suited for low-throughput targets rather than energy-optimal high performance.

Experiment 2 **confirms Hypothesis 2**: increasing Big CPU frequency yields large performance gains but incurs significantly higher power than the Little cluster, especially at high frequencies. The results highlight a governor-relevant operating strategy: the Big cluster should be activated only when required to satisfy throughput/latency, and when active, the governor should prefer moderate DVFS points that deliver most of the performance benefit while avoiding the inefficient top-frequency region.

3. Power-Performance graphs for AlexNet on GPU cluster at different Big/Little CPU frequencies.



Figuur 6: GPU-only: Power Consumption vs CPU Frequencies

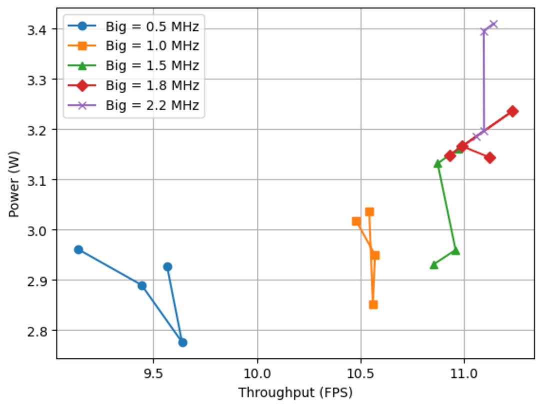
The 3 dimensional power plot shows that total system power increases approximately monotonically with both CPU frequencies, even though inference computation is performed on the GPU. This confirms that both CPU's remain active even when the GPU does the main work, reflecting CPU-side overhead workframe, such as scheduling, kernel launch, synchronization, and memory management. The slope is much bigger along the axis of the big CPU than along the axis of the little CPU. This implies that the most overhead is done by the big CPU. For practical application in the governor, this means that raising the big CPU frequencies has a much larger effect on the efficiency as GPU-only execution is much more invariant to little CPU changes.

In figuur 7, we presented Power vs Performance with grouping big CPU frequencies. Across the Big-frequency groups, FPS changes only modestly, while power shifts upward as Big frequency increases. This indicates throughput is GPU-dominated and the Big CPU frequency mostly contributes to baseline overhead power, not to major throughput gains. Same conclusion comes from the Power vs Latency graph (inverse graph).

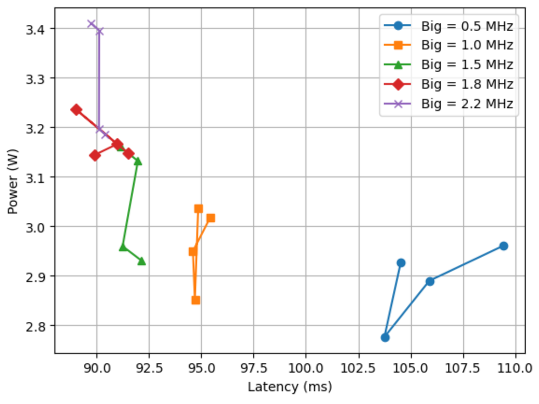
In Figuur 8, when little frequency are fixed, the grouped sets of measurements show a much stronger correlation between power and performance. This suggests that the little CPU cluster generally has less to do with the overhead of the pipeline.

Overall, the measurements lead us to the following conclusion: an efficient governor should treat CPU DVFS during GPU inference as a power-saving lever, and only increase CPU frequencies when latency constraints demand it.

Hypothesis 3 therefore be answered: CPU DVFS clearly affects power during GPU inference and has a measurable impact on performance, consistent with CPU-side orchestration

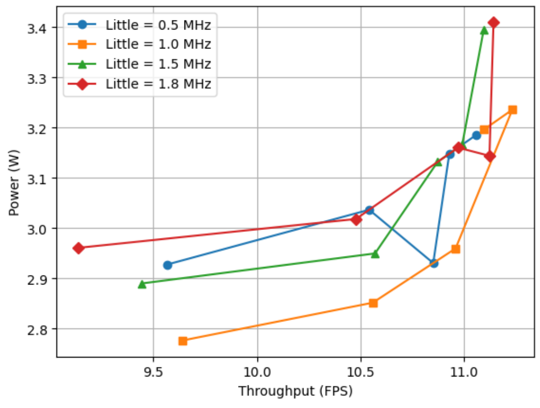


(a) GPU-only: Power vs Throughput

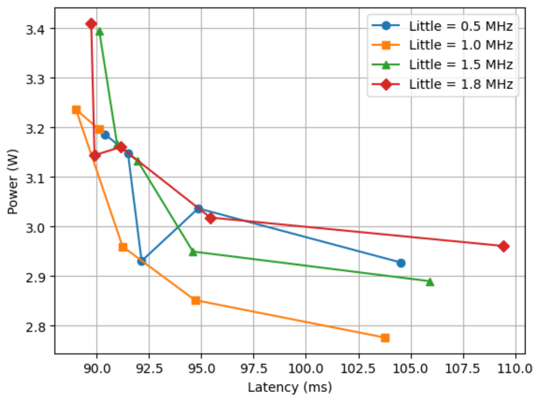


(b) GPU-only: Power vs Latency

Figuur 7: Power vs Performance graphs (fixed big cpu frequencies)



(a) GPU-only: Power vs Throughput



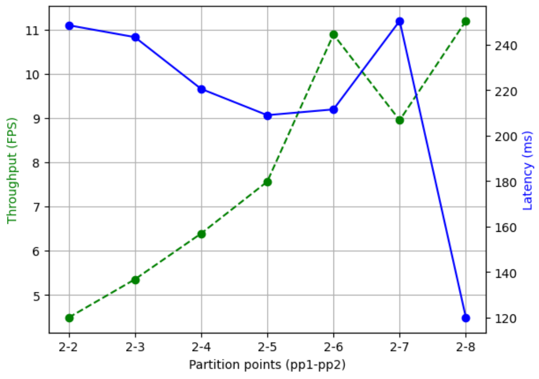
(b) GPU-only: Power vs Latency

Figuur 8: Power vs Performance graphs (fixed little cpu frequencies)

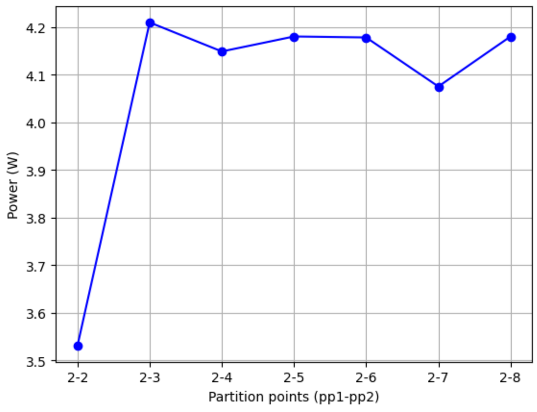
overhead. Moreover, we can conclude that the pipeline overhead is mainly done by the big GPU. This means, an increase of big CPU frequencies helps pipeline overhead much more than little CPU frequency increases. This should be respected in the design of the Governor.

4. Power-Performance graphs for AlexNet with fixed frequencies and component order and variable partition points.

For the 4. experiment with fixed GPU cluster size we got the results which are graphed out in Figuur 7:



(a) Partition Points vs Throughput (dashed) and Latency (solid)

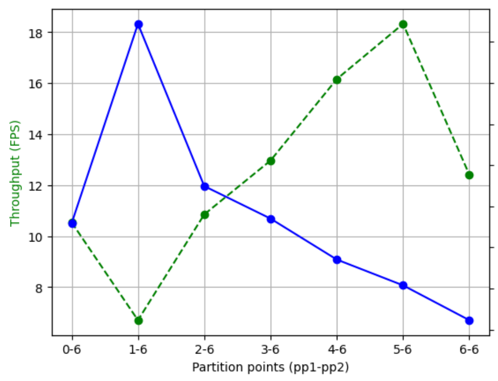


(b) Partition points vs Power

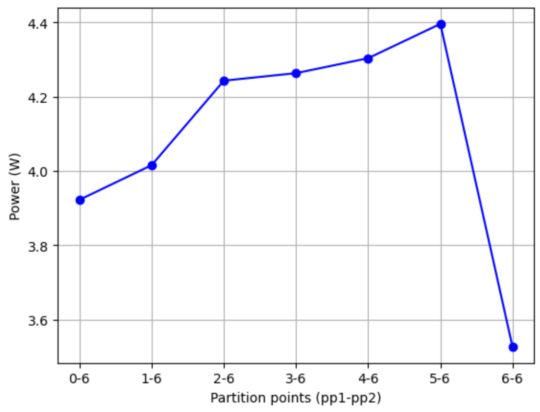
Figuur 9: Power-Performance graphs for variable partition points. GPU cluster fixed and CPU clusters variable.

The results of the first 4 measurements (partition points 2-2 to 2-5) are understandable and align with basic intuition. Performance increases, i.e. latency decreases and throughput increases, as the big CPU cluster takes on increasingly more workload and is more efficient. It is worth to mention that in the first measurement, power consumption is comparatively low as the big CPU cluster is excluded and therefore consumes only very limited amounts of energy.
The 5th, 6th and 7th measurement go against basic intuition. The expected behavior would be that the performance increasing trend continues. However, on the 5th measurement, latency decrease stalls while throughput still increases. The 6th measurement shows a not insignificant performance decrease. Finally, the last measurement records the best performance by far while power remains constant. This goes against intution, as the big CPU cluster takes increasingly more work load off the little CPU cluster, which should boost performance. Additionally, when the little CPU is completely excluded, the expectation is that performance drops because pipelines rely on multiple stages and there is less overall computing power that can be parallelized. The underlying problem that explains the counter-intuitive power and performance metrics during inference can be explained with the phenomenon of bottlenecks in pipelines. When the little CPU works on a small amount of layers it does very little useful work and adds more performance overhead as synchronization and memory management still occur. In other words, the overhead connected with the little CPU outweighs its computational performance. As soon as the little CPU is excluded this performance overhead is eliminated and the system works more efficiently. The slight dip in energy consumption on the partition points (2-7) proves that the system works less, i.e has longer idle times and therefore performs worse. To analyze such phenomena in more depth experiments with varying CPU frequencies would need to be ran.

The results of the set of measurements shown in Figuur 10 give insight into the tradeoffs between the little and big CPU during inference. Similarly, to the results of the first sub-experiment the results are mostly intuitive but also give insights into the underlying pipeline and bottlenecks. The first measurement shows pretty poor performance, which can be simply explained by the absence of the GPU. Obviously, the GPU contributes a lot to the computation of a neural network, especially in highly parallelizable computations. Especially in partition point 1-6, but also in 2-6 the bottleneck phenomenon of pipelines becomes evident again. The gain of using the GPU for the calculation of one layer is outweighed by the cost of pipeline overhead for reasons similar to those in the last experiment. Starting from measurement 4, the computational power of the GPU becomes evident as it outweighs the pipeline overhead and the performance monotonically increases. When the large CPU is completely excluded, a drop in performance can



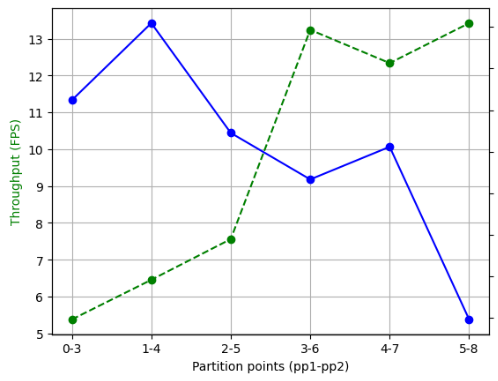
(a) Partition Points vs Throughput (dashed) and Latency (solid)



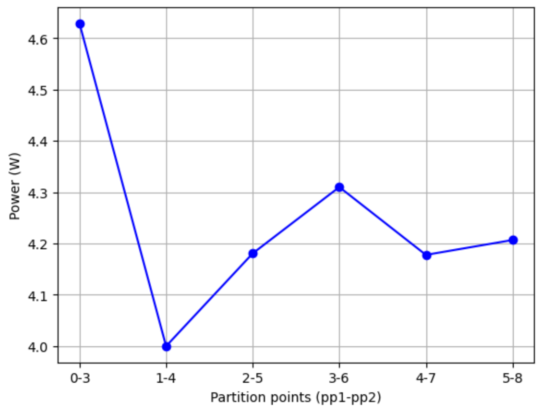
(b) Partition points vs Power

Figuur 10: Power-Performance graphs for variable partition points. Little CPU cluster fixed and big CPU and GPU clusters variable.

be observed in the last measurement, as the efficiency of the large CPU cluster cannot be compensated for by the GPU. The short big CPU stage does not cause a drop in performance as its computational power outweighs the pipeline overhead. The power consumption is in line with basic intuition as it increases with increasing GPU computation and abruptly drops when the big CPU cluster is excluded.



(a) Partition Points vs Throughput (dashed) and Latency (solid)



(b) Partition Points vs Power

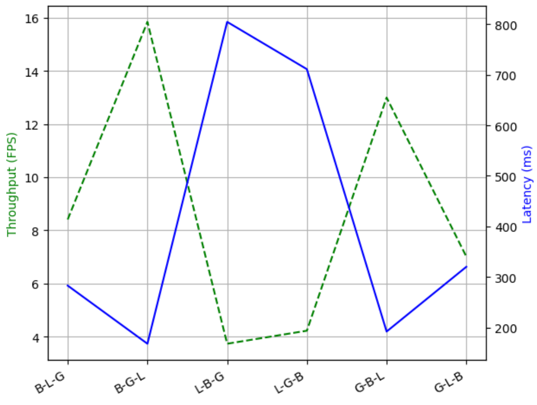
Figuur 11: Power-Performance graphs for variable partition points. Big CPU cluster fixed and little CPU and GPU clusters variable.

The results of the last sub-experiment shown in Figuur 11 show the power-performance analysis of a fixed big CPU cluster and the trade off between little CPU and GPU. As the big CPU is the middle stage of the pipeline the majority of the findings are similar to the ones already made in the previous sub experiments. The performance is bad at the beginning as the GPU is excluded. With the GPU only taking on the first layer the latency increases more because of the reasons explained above. The reason why the performance decrease is not as notable as in the previous experiment is that there are more layers assigned to the little CPU cluster. The bad performance of the slow little CPU cluster therefore overshadows the bottleneck of the short GPU stage. Similarly to the first experiment, the performance then increases monotonically until the bottleneck of the short little CPU stage is hit. Finally, the performance peaks as the GPU and big CPU work well together in a two stage pipeline. The power is the highest on the first measurement because the long little CPU stage is extremely inefficient. In the following measurements the power remains constant around the 4.2W level.

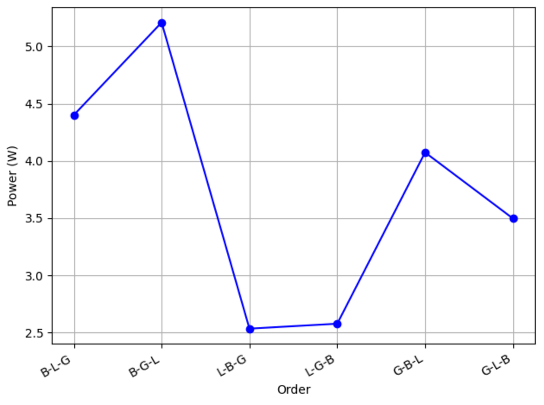
The findings of this experiment tie in with the hypothesis. We found that indeed early convolutional layers of AlexNet are computationally intensive and should mainly be assigned to the GPU. Assigning too few layers to the GPU massively underutilizes it and leads to bad performance and efficiency. Similarly for the big CPU cluster, the workload should match high the computational capacities. Unexpectedly, stages that have too few layers, even for the little CPU cluster, caused more harm than good as the pipeline overhead got evident. For even better insight this experiment needs to be repeated with different frequencies and pipelines of only 2 stages. The overall learning for the development of a governor is that

5. Power-Performance graphs for AlexNet with fixed frequencies and partition points and variable component order.

The 5th experiment explored the component order of the pipeline with everything else fixed. The results can be be seen in Figuur 10:



(a) Order vs Throughput (dashed) and Latency (solid)



(b) Order vs Power

Figuur 12: Power-Performance graphs for inference on AlexNet with variable component order.

From the 6 measurements a clear result can be formulated. It is evident that the orders with the worst power-performances always start with the little GPU cluster. This is in line with intuition, as the little CPU is the worst fit to execute many parallel computations which mainly compromise the convolutional layers of AlexNet. Furthermore, for similar reasoning, even pipelines where the little CPU takes on the middle stage, perform very poorly. This effectively renders 4 permutations of component order useless, as the same performances can be reached much more efficiently. More interesting are the component orders B-G-L and G-B-L. The two orders that assign the most powerful assets of the Amlogic A311D system to the most computationally intensive layers of AlexNet obviously perform the best. Fairly surprisingly however, the GPU-first order does not beat the big CPU-first order in performance. We expected that the GPU-first approach wins against the big CPU-first approach for obvious reasons. The reason why it can't can probably be found in the other parameters of the system - especially selected frequencies and partition points (which were 2-5) in this experiment. However, regardless of its lower performance the G-B-L order does indeed outperform B-G-L in efficiency, i.e. the ratio between power consumption and performance. This aligns with the intuition and our hypothesis. Most probably, if other parameters in this system were chosen differently, G-B-L would also outperform B-G-L in sheer performance.

This finding was the most significant. We observed how the order is the most vital part for a well functioning inference pipeline. The hypothesis "to assign computationally heavier stages to faster processing elements [...] earlier in the pipeline [to] achieve higher throughput and lower latency"was confirmed. We conclude that for the development of the governor the only possible order to achieve an efficient solution is G-B-L and no plausible trade-offs can be made with other component orders.

5 Conclusion [0.25 Points]

This report characterized the power–performance behavior of AlexNet inference on the Khadas VIM3 across CPU DVFS scaling, GPU execution with varying CPU frequencies, and Pipe-All pipeline knobs (partition points and component order). The CPU-only results show the expected DVFS trade-off: higher frequency increases throughput and reduces latency, but with diminishing returns at the top frequencies. The little cluster provides gentle, predictable scaling suited for low-performance targets, whereas the big cluster delivers much higher performance at a substantially higher power cost. Efficiency (FPS/W) highlights a mid-frequency "sweet spot" where most performance benefit is obtained without disproportionate power overhead. For GPU-only execution, throughput is largely GPU-dominated, but system power increases noticeably with CPU DVFS (especially big), confirming non-negligible CPU-side orchestration overhead. Finally, partitioning and ordering experiments demonstrate that pipeline overhead and bottlenecks can dominate: very small stages on slow components can reduce overall performance despite offloading computation, while orders that map early, compute-heavy layers to GPU/Big CPU produce the best power–performance behavior.

The main underlying problem that we had when designing the experiments, especially the 4th and 5th experiment is the unbalanced distributions of weights across AlexNet. Since the convolutional layers are much more computationally expensive and large than the final fully connected layers of AlexNet the weight distribution is not straight forward. That means that some of our claims might be based on wrong assumptions and therefore not very representative for the development of the Governor. For instance in Experiment 4, when choosing a partition point for a reasonable size of the GPU stage, the choice of layer 2 or 3 is of vital importance, however, we weren't able to confirm that our assumptions were the best for the underlying model.

The findings from experiments directly influence the design of our governor. For CPU execution, the governor should avoid top DVFS points and instead operate near the observed efficiency "sweet spots," selecting Big cores only when throughput/latency constraints cannot be met by Little. For GPU execution, CPU DVFS should be treated primarily as a power-saving knob, increasing CPU frequency only when latency degradation indicates CPU-side overhead becoming limiting. For Pipe-All scheduling, the governor should avoid configurations that create very short stages on slow components and should prefer component orders and partition points that balance pipeline stages to prevent bottlenecks.

6 Mandatory Appendix

Should the grade be divided equally between all team members: **Yes, we both worked on everything together**.

Explain individual contributions (roles) of all team members.

First we discussed and designed the experiments together. Matteo did most of the measuring and collected the results, I plotted most of the graphs. And then we gradually wrote the report together, section by section and cross read each others sections to get a grasp of the others reasoning and follow the 4-eye-principle.