Team 9 Governor Report

Levi van der Griendt, Anthony Xuan Pham

January 28, 2023

1 Introduction of Design

Our governor design report delves into the intricacies of creating an algorithm that effectively balances performance and efficiency. The governor, at its core, operates through two distinct yet complementary methods. The first method involves the utilization of a binary search algorithm to pinpoint a solution that is both optimal and efficient. This is achieved by sampling a set of points and subjecting them to the binary search process.

The second method, on the other hand, focuses on the efficient management of load. To achieve this, we have implemented a binary tree structure that organizes elements based on their performance and power consumption. This provides us with a solid foundation for the second and more intriguing aspect of our algorithm.

To gather the necessary data, we have analyzed a comprehensive power performance report of CNNs and have used this information to construct a power module. This module employs a simple cubic regression technique to provide a better understanding of the relationship between different power settings.

Once the optimal frequencies have been identified, we proceed to the load balancing stage. This is arguably the most crucial and fascinating aspect of our governor. The load balancer has two primary objectives: to lower the frequency and to prevent performance from dropping. To accomplish this, we have developed a performance model that utilizes the data collected earlier and predicts the impact of frequency changes on performance.

Finally, we address the issue of order. To determine the most effective sequence, we have conducted a series of tests and have chosen the order that yields the best results. However, it must be noted that any deviation from this order would significantly expand the search space and impede the governor's ability to find a satisfactory solution in a reasonable time frame. While this may be considered a limitation of our governor, it is a trade-off we have deemed necessary in order to achieve optimal performance.

Overall, our governor design report presents a comprehensive and in-depth examination of the intricacies of balancing performance and efficiency. Through the utilization of advanced techniques such as binary search, cubic regression, and performance modeling, we have created a powerful algorithm that effectively manages load while maintaining optimal performance.

2 Improvements following Power-Performance Characterization Observations

In our power-performance characterization observations, we identified three key areas for improvement in the original governor design for our system.

Firstly, we found a logarithmic relationship between the power usage of our two CPUs and the dependent variables of target FPS and latency. This allowed us to develop two models: one for the relationship between frequencies and FPS/latency, and another for the relationship between FPS/latency

and power usage. Using these models, we developed a program to select the optimal frequencies for the two CPUs that achieve the desired level of FPS and latency, while minimizing power usage. We also implemented a method to partition certain parts of the CNN among the processors, which further optimized the power usage.

Secondly, we discovered that the order of the processors also had an impact on the performance of the system. While the impact was not as significant as the levels of frequencies, it still played a role in determining the final power usage. We also found that choosing the wrong order of processors could lead to suboptimal results. To address this, we included the order of processors in our optimization procedure to find the optimal order that results in the desired level of latency and FPS, while minimizing power usage.

Thirdly, we found that the proportion of inference time for each processor in a particular order was influenced by the levels of maximum frequencies used by the Big and Little CPUs. To further investigate this, we combined and followed the processes of our two initial experiments: changing frequencies at a certain order and changing order at constant frequencies for the CPUs. This revealed that the frequencies heavily influenced the power usage of the Big and Little CPUs and changed the total amount of time taken by the processors. This prompted us to implement a load balancing technique to distribute the workload evenly among the processors, so that the amount of inference times for each processor is approximately identical at a particular combination of frequencies and order of the processors. This not only maximizes FPS at a certain level of frequencies and order, but also minimizes the amount of delay time for the bottleneck processor and ensures a minimized amount of total input time for the three processors.

Overall, our power-performance characterization observations have allowed us to improve the original governor design in three key ways: optimizing the levels of frequencies, optimizing the order of processors, and implementing a load balancing technique to distribute the workload evenly among the processors. By implementing these changes, we are able to achieve the desired level of FPS and latency, while minimizing power usage. This not only improves the performance of our system but also reduces energy consumption, which is crucial in today's world of increasing energy costs and concerns about environmental sustainability.

2.1 Current Test Model

In order to provide an in-depth analysis of the efficacy of our Governor algorithm, we conducted a series of tests utilizing a simplified version of both the search tree and load balancer components. While the search tree component was not fully optimized in these tests, the load balancer was designed to encompass the majority of crucial considerations. However, as we delved deeper into the results of the tests, it became apparent that there are areas for improvement in the load balancer, particularly in regards to avoiding reaching programming local minimums. Additionally, the current version of the load balancer is relatively simplistic and does not actively change the frequency, instead focusing on maximizing performance within the current frequency parameters.

Furthermore, the test model we used has not yet incorporated a fully fleshed-out design for the selection of processor orders. Currently, the orders are arranged in a two-dimensional array, with the worst-performing processors listed first and the best-performing listed last. The middle processor is then chosen as the starting point for the algorithm. While this method may be sufficient in the short term, further research and development is needed to ensure that the Governor algorithm is able to select the optimal processor order for a given task.

In summary, our tests have revealed that while the Governor algorithm shows promise, there are several areas that require further optimization and refinement before it can be considered fully operational. The search tree and load balancer components, in particular, need to be fine-tuned to ensure optimal performance, and a more sophisticated approach to the selection of processor orders is necessary to truly harness the full potential of the algorithm.

3 Method

In order to design an effective governor for our system, we employed a two-pronged approach. The first step was to utilize a binary search algorithm to identify a point close to a solution that is both efficient and sufficient. To accomplish this, we sampled a number of points and ran them through the binary search process. The second step in our governor design was to implement an efficient load balancing mechanism. To achieve this, we utilized a binary tree to order our data points based on their performance and power usage. This allowed us to establish a starting point for the load balancing process. To gather data for our governor, we collected values for every CNN power performance rating in our power report. From this data, we constructed a power module with a simple cubic regression algorithm to provide a more accurate scale between different power settings. Once the optimal frequencies were established, we turned our attention to the load balancing process. The load balancer is a crucial component of our governor, and its primary goal is to lower the frequency while maintaining optimal performance. To accomplish this, we used a performance model with the same data set as before to predict any potential performance drops at a given frequency. In terms of order, we again used data collected from our tests to determine the best order for our processors. Our current design is a 2-dimensional array, where we arranged the processors from worst- to best-performant, and then selected the middle one as the starting order. To test the effectiveness of our governor, we conducted several experiments using a basic version of both the search tree and the load balancer. The search tree was not fully optimized and the load balancer was designed to cover the most important aspects, but there is still room for improvement. The current test model still has not implemented a sensible enough design for the choice of orders of processors. Overall, the results of the tests showed that our governor performed well against the demo governor, but there are still areas that can be improved. While we did reach a local optimization point for the power usage at the target FPS and latency using this method, it also served as an important step for us to identify areas for improvement in our design. The main focus of the tests was to compare the algorithmic designs of the 2 governors when both of them only use 1 particular order. This allows us to understand much deeper about the frequencies optimization design and the current load balancing algorithm, to improve them further in the upcoming week.

4 Results

The results of the quantitative performance comparison between the example governor and our demo governor are presented in this section. Three tests were conducted, each with a different system (CNN), and target frames per second (FPS) and latency goals. The example governor and demo governor were given the same amount of time (7, 10, and 6 minutes respectively for the 3 tests) to output a design point that maximized the frequencies used by the Big CPU and Little CPU, as well as the locations of the two partitioning points.

System	Target FPS & Latency	Governor result	Governor time	Demo Governor result
AlexNet	7 & 300	2100000 & 180000 part1: 3 part2: 6	6:43	2208000 & 1800000 part1:2 part2: 4
GoogleNet	4 & 400	1908000 & 1608000 part1:3 &part2:10	9:34	2208000 & 180000 part1:4 &part2:6
RestNet50	3 & 900	2208000 & 1800000 part1:9 part2:15	5:37	22080000 & 1800000 part1:4 part2:9

In the first test, the system used was AlexNet, with a target FPS of 7 and a latency goal of 300ms. The example governor's design point was 2100000kHz for the Big CPU and 180000kHz for the Little CPU, with the first partitioning point at the 3rd part of the CNN and the second partitioning point at

the 6th part of the CNN. The demo governor, on the other hand, reached a design point of 2208000kHz for the Big CPU and 1800000kHz for the Little CPU, with the first partitioning point at the 2nd part of the CNN and the second partitioning point at the 4th part of the CNN. This indicates that our demo governor was able to achieve higher frequencies for both the Big and Little CPUs, as well as a more optimized partitioning of the CNN.

In the second test, the system used was GoogleNet, with a target FPS of 4 and a latency goal of 400ms. The example governor's design point was 1908000kHz for the Big CPU and 1608000kHz for the Little CPU, with the first partitioning point at the 3rd part of the CNN and the second partitioning point at the 10th part of the CNN. The demo governor, on the other hand, reached a design point of 2208000kHz for the Big CPU and 180000kHz for the Little CPU, with the first partitioning point at the 4th part of the CNN and the second partitioning point at the 6th part of the CNN. Once again, our demo governor was able to achieve higher frequencies for both the Big and Little CPUs, as well as a more optimized partitioning of the CNN.

In the third test, the system used was RestNet50, with a target FPS of 3 and a latency goal of 900ms. The example governor's design point was 2208000kHz for the Big CPU and 1800000kHz for the Little CPU, with the first partitioning point at the 9th part of the CNN and the second partitioning point at the 15th part of the CNN. The demo governor, on the other hand, reached a design point of 22080000kHz for the Big CPU and 1800000kHz for the Little CPU, with the first partitioning point at 9th part of the CNN and the second partitioning point at the 15th part of the CNN. In this test, the demo governor was able to achieve the same frequencies for both the Big and Little CPUs as the example governor, but was able to maintain the same partitioning point.

Overall, the results of the quantitative performance comparison indicate that our demo governor is able to achieve higher frequencies for both the Big and Little CPUs, as well as more optimized partitioning of the CNNs in comparison to the example governor. Additionally, our demo governor was able to achieve the same frequencies and partitioning as the example governor in the case of RestNet50, while still maintaining the same partitioning point. These results demonstrate the effectiveness of our demo governor in maximizing the performance of the system while meeting the target FPS and latency goals.

It is important to note that these results were obtained by using specific test case, and it's important to test the demo governor with a diverse set of neural networks and use cases to validate its generalization to the solution set. We also acknowledge that there is room for improvement in the design of the order of processors, as well as optimization of the search tree and load balancer. Further analysis of the results will be necessary to determine the specific areas for improvement and to develop a strategy for implementing those changes.

5 Conclusion

Overall, our demo governor demonstrated an ability to achieve equal or better results compared to the example governor. However, it should be noted that there was not a significant improvement in performance. This could be due to the fact that our governor makes too many choices during the process, causing it to engage in load balancing at multiple points rather than later on in the process. This highlights the need for a smaller decision tree to make choices more efficiently.

Additionally, while the load balancing feature of our governor appears to be working, it is difficult to draw any concrete conclusions about its efficiency based on the data presented. Further testing and analysis would be needed to fully evaluate the effectiveness of this feature.

In conclusion, our demo governor showed promise in its ability to achieve better results compared to the example governor. However, there is still room for improvement, specifically in the area of decision-making and load balancing. Further research and development is needed to fully optimize the performance of our governor. To improve it, we need to work on the decision tree which will help to reduce the number of choices made during the process and make the load balancer more efficient.