Optimization of Automotive Software Distribution on Multi-core Systems using Machine Learning Approaches

Syed Aoun Raza, Amal Jose Vallavanthara, Robert Bosch GmbH, Rakesh Nidavani, Robert Bosch Engineering & Business Solutions;

1 Abstract

Multi-core software should be partitioned under different constraints e.g., balanced execution load on cores, timing behavior and optimized level of communication/synchronization among different system components. The objective is to efficiently distribute the processes onto multi-core hardware such that the system has reduced communication/synchronization complexity. Moreover, a bad distribution strategy during migration from single- to multi-core and from multi- to many-core hardware does not always return the expected performance gain. This paper presents two novel AI-based approaches for optimal distribution (minimal inter-core communication inspite of no deadline misses) of software system on multi-core hardware architecture. We discuss the comparisons of our machine learning solutions based on unsupervised and reinforcement learning. We share the benefits and limitations of using unsupervised learning and reinforcement learning based on our experience.

2 Introduction

Moore's law is the observation that the density of transistors in an integrated circuit would double every year. However, over the last two decades, chips have reached the epitome of the density of transistors and growth rate is slowing down. Therefore, engineers moved to multi-core processors and microcontrollers to keep up with the pace of industry requirements. However, multi-core systems would turn out to be a disaster if the software is not distributed efficiently among the processors. To achieve maximum gain on multi-core hardware work-load distribution strategy plays a significant role. This provides a guideline on how to fullfill system constraints and keep the factors to a minimum which generate overhead and reduce the overall system gain e.g., keeping the processes on same core which have maximum communication in the system. Considering, the complexity of hardware and software machine learning methodologies can support multi-core system architects to devise an optimal distribution strategy.

3 State of the art

Migration of multi-core automotive embedded system is challenging [1]. Through AUTOSAR application-level architecture distribution, Engineers at VALEO [2], have achieved 47% optimization w.r.t inter-core communication. Machine learning algorithms have been used previously for multi-core optimization in different fields to optimize constraints like power consumption [3], parallelism [4], etc. However, scholarly articles on using machine learning for automotive software process level distribution to reduce inter-core communication on automotive ECUs are not yet openly available or have not been studied.

4 Data Preparation

Usually, an automotive software contains several million lines of code and artifacts and distributing them optimally is beyond naive human capability. Therefore, we use a model based approach to convert system information from different artifacts into an AMALTHEA model. Our decision to use AMALTHEA model is based on [5], which describes how AMALTHEA model addresses some of the deficits of AUTOSAR especially in multi-core systems. We analyze the task, interrupt, runnable and data interactions to record communications frequency among them. This data is analyzed to create a communication model of the system. Initial process to core mapping was based on the communication model and computed process loads.

5 Why unsupervised learning?

Machine Learning is classified into supervised learning, unsupervised learning and reinforcement learning. Supervised learning is the training technique where input data is labeled. Unsupervised learning is a technique where data is not labeled and the system tries to self learn from the data and its patterns. In reinforcement learning, the system is placed in an environment wherein it produces its own input data and then trains itself using the generated output. It is difficult to use supervised learning for automotive software distribution because of the following reasons. Firstly, the data model is huge and disparate. Secondly, labeling the automotive software data is a complex task and finally, limited availability of optimally distributed architecture makes it difficult to train a sustainable model. Therefore unsupervised learning is a convenient approach when only few parameters need to be considered. Initially, the task in hand was to cluster the processes and map the resultant clusters to cores based on CPU resource utilization and inter-process communication alone. We used a modified K-Means algorithm [6] to cluster the highly communicating processes together. The algorithm provides

an elegant solution to cluster sparsely connected processes. The algorithm could easily identify and cluster processes which highly communicate with each other. For instance, it will cluster processes into autonomous driving processes, engine control processes, car multimedia processes, etc, since the tasks in a component have more probability of communicating with each other. The software architect could also find the interfaces between these functionalities.

6 Why Reinforcement learning?

Distributing a dense software architecture performing a single functionality across a multi-core microcontroller requires a different approach. In automotive software, the highly dependent engine control processes (e.g. the periodic-control-tasks) communicate frequently with each other. The K-Means clustering algorithm grouped all these high-load processes and assigned them to one core and a few low-load processes on other cores, thus not achieving optimal load balancing. The algorithm is better suitable for reducing the inter-core dependency in sparsely connected processes. Dissecting a densely connected cluster of processes into smaller clusters would be more optimal if we consider more parameters like processes' timing behavior, core affinity, etc. The quality of the predicted model deteriorates with increase in number of considered parameters. Unsupervised learning comes with drawbacks like inflexibility and requires more development effort. For instance, addition of new constraints like core affinity, memory usage or priorities would require re-work of the algorithm as a whole that leads to high development effort and slow delivery to market. Moreover, the lack of traceability and explainable AI in the K-Means unsupervised learning makes it difficult to point out the exact reasoning for the decisions made. We required more flexibility, traceability and shorter delivery time. Reinforcement learning was the next approach we successfully tried to partition densely connected processes by incorporating more constraints. As shown in figure.1, our reinforcement-learning algorithm consists of an interpreter, agent and an environment.

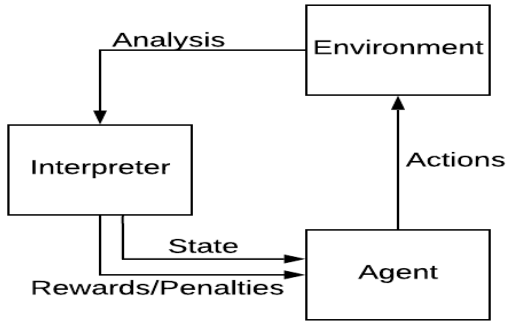

Figure 1: Reinforcement learning Work-Flow

The interpreter evaluates the environment and awards rewards and penalties for actions. The software agent uses these rewards and penalties to distribute the architecture. As a large number of machine-states need to be evaluated, the training takes an ample amount of time. This was a major drawback. The trick to overcome this is to extract only essential information from the AMALTHEA model to replicate the original multi-core environment.

7 Summary

In this paper, we discussed the distribution of multi-core software architecture using machine learning approaches. We revealed our recipe for data preparation to consolidate the large quantity of data to minimal amount for efficient data processing. We gave an insight into the benefits and consequences of using unsupervised learning and reinforcement learning. We reveal how we were able to build a solution that provides the user the power to distribute the tasks among the cores by taking into consideration the following parameters:1. Load of each core, 2. Load of each process, 3. Communication of each task, 4. Amount of label and runnable accesses, 5. Execution time of the task such that the task deadlines are not missed, 6. Constraints set by the users.

In future, we plan to incorporate deadlock and consistency checks in reinforcement learning to make decisions on assignment of processes to cores. The test results showed up to 300% optimization in intra core communication by using this method.

References

[1] Leteinturier, P., Brewerton, S., and Scheibert, K., "MultiCore Benefits & Challenges for Automotive Applications"

[2] Wenhao Wang, Sylvain Cotard, Fabrice Gravez, Yael Chambrin, Benoit Miramond. Optimizing Application Distribution on Multi-Core Systems within AUTOSAR.

[3] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," 2015 Design

[4] Zheng Wang and Michael F.P. O'Boyle. 2010. Partitioning streaming parallelism for multi-cores: a machine learning based approach.

[5] Andreas Sailer, Stefan Schmidhuber, Maximilian Hempe, Michael Deubzer and Jurgen Mottok, A Practical Comparison of ASAM MDX vs. AUTOSAR vs. AMALTHEA

[6] S. Geetha, G. Poonthalir, T. Vanathi, "Improved K-Means Algorithm for Capacitated Clustering Problem"