The principles of building a machine-learning-based service for converting sequential code into parallel code

Ivan Viktorov and Ruslan Gibadullin

Abstract. This article presents a novel approach for automating the parallelization of programming code using machine learning. The approach centers on a two-phase algorithm, incorporating a training phase and a transformation phase. In the training phase, a neural network is trained using data in the form of Abstract Syntax Trees, with Word2Vec being employed as the primary model for converting the syntax tree into numerical arrays. The choice of Word2Vec is attributed to its efficacy in encoding words with less reliance on context, compared to other natural language processing models such as GloVe and FastText. During the transformation phase, the trained model is applied to new sequential code, transforming it into parallel programming code. The article discusses in detail the mechanisms behind the algorithm, the rationale for the selection of Word2Vec, and the subsequent processing of code data. This methodology introduces an intelligent, automated system capable of understanding and optimizing the syntactic and semantic structures of code for parallel computing environments. The article is relevant for researchers and practitioners seeking to enhance code optimization techniques through the integration of machine learning models.

1 Introduction

In contemporary computing environments, there has been a marked shift in the approach to enhancing processor performance due to the stagnation in clock frequency scaling. This stagnation is often attributed to limitations in power and heat dissipation. Consequently, the evolution of performance improvement has pivoted toward the incorporation of an increasing number of processor cores, known as multicore processors. This development has resulted in the growing importance and relevance of multithreaded programming paradigms, as leveraging multiple cores typically involves the concurrent execution of threads [1, 2].

However, transitioning to multithreaded programming introduces a set of challenges:

- Complexity: multithreaded programming is an intrinsically intricate domain with a plethora of subtleties. Effective exploitation of concurrency necessitates highly skilled programmers who possess the adeptness to adeptly transform sequential code for execution on multicore processors.

* Corresponding author: landwatersun@mail.ru
processors. This often involves dealing with issues such as synchronization, data races, and thread safety.

- Additional Costs: the development of multithreaded applications entails additional overheads for companies. Beyond the need for skilled programmers, companies must also allocate resources for more qualified analysts and testers. This is necessary to ensure that concurrent code maintains its integrity and correctness under various threading scenarios.

One potential solution to alleviate these challenges is the development of a translator capable of transforming sequential code into parallel code. This would essentially automate the process of parallelization, thus reducing the need for specialized human intervention.

At present, fully-fledged solutions for automatic parallelization are not available. This is attributable to several factors, including the prevalence of nested functions and complex loop constructs in code, which complicates the analysis of data dependencies. There are, however, several tools available on the market that employ a partial parallelization approach. Examples of such tools include Polaris, Pluto, Par4All, WPP, SUIF, VAST/PARALLEL, OSCAR, Parawise, Cetus, CAPO, and SAPFOR. These tools typically operate on similar principles, but they also share a set of limitations including dependency on the target platform, the necessity to keep track of software updates, and the lack of support for full automation in parallelization [3, 4].

As a prospective resolution to this conundrum, one could envisage the creation of a translator based on a web application that utilizes a code transformation service powered by machine learning. This would harness the capability of machine learning algorithms to analyze code structures and dependencies, and consequently facilitate more efficient and automated code parallelization. This approach could also be platform-agnostic and continuously adaptive to changes in programming paradigms and hardware architectures.

## 2 Translator architecture

Figure 1 depicts the architecture of a software system designed to transform sequential code into parallel code. This architecture is based on the microservices pattern and is capable of supporting various programming languages.

The software system comprises several microservices:

1) Request Processing Service
2) Syntax Tree Construction Service
3) Code Transformation Service

The Request Processing Service is responsible for handling REST API requests from users. These requests can include the conversion of sequential code into parallel code, retrieval of transformed code, and retrieval of information about the syntax tree. The requests contain the sequential code that needs to be parallelized, and the programming language in which the code is written [5, 6].

The Syntax Tree Construction Service converts the received sequential code into an abstract syntax tree representation. The abstract syntax tree is a hierarchical representation that describes the structure of the source code. The service stores the data about the syntax tree into a database. Upon successfully storing the syntax tree data, it sends an operation identifier to a message broker, signaling the completion of the syntax tree construction and indicating that the Code Transformation Service can commence [7, 8].

The Code Transformation Service retrieves the syntax tree data from the database using the operation identifier received from the message broker. It then employs various parallelization strategies to construct different parallelization schemes for the original program. This involves analyzing the abstract syntax tree according to a set of rules retrieved from the database that correspond to the syntax of the programming language. The transformation of code from the syntax tree representation may be performed using methods
like the polyhedral model, which is a mathematical representation used for optimizing loop nests in terms of execution time and parallelism. The service generates the text of the program utilizing principles of parallel programming, and the transformed code is stored back in the database for retrieval [9, 10].

Additionally, machine learning can be employed within the Code Transformation Service to facilitate the code transformation process. Through the use of machine learning algorithms, the service can learn from existing patterns and potentially discover more efficient ways of parallelizing the code.

This microservices-based architecture allows for modularization and scalability, making it possible to extend support for additional programming languages and implement new transformation techniques as they emerge. It also permits components to be updated or replaced independently, thereby increasing the maintainability of the system.

![Translator architecture using message broker](image_url)

Fig. 1. Translator architecture using message broker.

The most suitable architecture for constructing a neural network for a code transformation service encompasses the Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Long Short-Term Memory networks (as shown in Figure 2 and Figure 3).
The Multilayer Perceptron is a type of artificial neural network characterized by its feedforward structure. A distinctive feature of the MLP is that it consists of at least three layers – an input layer, one or more hidden layers, and an output layer. The MLP employs backpropagation as its learning algorithm, which involves computing the gradient of the loss function concerning the weights for a single input-output example and updating the weights using gradient descent. Non-linear activation functions are utilized in the neurons of the hidden and output layers, enabling the MLP to model complex, non-linear relationships in the data. In the MLP architecture, neurons in adjacent layers are fully connected, following an “all-to-all” scheme. As of now, MLP serves as one of the foundational elements in artificial neural networks [11, 12].

Convolutional Neural Networks, illustrated in Figure 3, have an architecture specifically tailored for pattern recognition within images and are built based on principles of visual perception. A key feature of CNNs is the use of convolutional layers, where the neurons apply a convolution operation to the input, passing the result to the next layer. This is followed by pooling (or subsampling), which reduces the spatial dimensions of the feature maps. The convolution operation allows the network to focus on local features, while successive layers can learn to recognize higher-order features. In convolution, fragments of input data are multiplied by a convolution matrix (kernel), resulting in feature maps. Typically, multiple kernels are employed to detect various features. The pooling operation serves to downsample the feature maps and reduce computational complexity.

An alternative approach for code transformation can be based on the Long Short-Term Memory network, a specialized type of Recurrent Neural Network. LSTM networks are particularly well-suited for sequence prediction problems, owing to their capability to store past information. This is crucial for tasks where context and order are important. LSTMs have memory cells that allow them to maintain and retrieve information over long sequences, making them capable of understanding context and dependencies in the input data. The memory in LSTM is short-term and mutable; during training, information in the memory mixes with new input and eventually gets overwritten after several iterations [13, 14].

For a code transformation service, utilizing a combination of these neural network architectures can be highly effective. For instance, CNNs could be employed for feature extraction, while LSTM networks can be used to maintain context and dependencies. Meanwhile, MLP could serve as the final fully connected layers for generating output in the required format. This kind of hybrid architecture harnesses the strengths of each neural network type, thereby leading to more accurate and effective code transformation.

*Fig. 2. Multilayer perceptron architecture.*
3 Algorithm

The algorithm for the parallelization of a program based on machine learning can be segmented into two modes. The first mode is the training phase, and the second mode is the transformation phase, which converts sequential code into parallel programming code.

For transforming code represented as an Abstract Syntax Tree (AST) into numerical values, it is necessary to employ models that process natural language. To address this task, one could either develop a custom Recurrent Neural Network Language Model, which may be deemed as a redundant solution, or employ a pre-existing model [15, 16].

As of now, there are several models available for natural language processing which include Word2Vec, GloVe, and FastText. Word2Vec is predicated on the hypothesis that words which frequently occur in similar contexts tend to have analogous meanings. On the other hand, GloVe is predicated on minimizing the discrepancy between the product of word vectors and the logarithm of the probability of their co-occurrence, using stochastic gradient descent. FastText, an extension of Word2Vec, was introduced by Facebook in 2016 and distinguishes itself by breaking words into subwords or n-grams before feeding them into the neural network, instead of treating each word as a whole. Among these three models, Word2Vec is generally more preferred since it doesn’t rely on the context of the word, unlike GloVe and FastText.

In Figure 4, the algorithm for code parallelization is illustrated for a code transformation service based on machine learning. Consider the basic steps of the algorithm.

1) Data, including code identifiers, are received from a message broker or an AST construction service.
2) The code data in the form of an Abstract Syntax Tree is retrieved from a database using the code identifier.
3) Iterative portions of the code are extracted in the form of an Abstract Syntax Tree.
4) Using Word2Vec, the extracted portion of the Abstract Syntax Tree is converted into a numerical array.
5) A training flag is extracted, which is set when the parallel program is adjusted by a programmer.
6) In training mode, the network is trained with new data.
7) In the absence of the training mode, the transformed parallel code is extracted from the neural network.
8) As a final step, the numerical array is converted back into syntactic constructs of the programming language, and a parallelized code program is assembled.
Conclusion

In conclusion, the efficient parallelization of programs is an essential aspect of optimizing computing resources and enhancing performance. The presented approach employs machine learning models to automate the transformation of sequential code into parallel code. The algorithm comprises two main modes, the training phase and the transformation phase.

During the training phase, the neural network is trained with data representing code in the form of Abstract Syntax Trees. Word2Vec is used as a primary model to convert the syntax tree into numerical arrays because of its property to encode words without relying too much on context, making it more suitable compared to other models like GloVe and FastText.

The transformation phase involves the application of the trained model to new sequential codes for converting them into parallel programming code. This is achieved by processing the code data, extracting the iterative portions, and using the trained neural network to generate a parallelized version of the input code.

The use of machine learning in code parallelization introduces an automated and intelligent system that can understand the syntactic and semantic structure of code. This system can considerably expedite the process of code optimization for parallel computing environments.

It's noteworthy that the choice of the natural language processing model and the neural network architecture are critical to the performance and accuracy of the parallelized code.
Moreover, continuous training and adjustment by programmers can ensure that the model stays relevant and adapts to evolving programming paradigms.

Overall, leveraging machine learning models for code parallelization promises a streamlined and more effective approach to optimizing programs for parallel processing. This contributes not only to the utilization of computational resources but also to the advancement of automated code optimization techniques. Future research in this area may explore the integration of more advanced natural language processing models and neural networks, as well as the application of this methodology across various programming languages and computing platforms.

References

8. K. Kulagin, M. Salikhov, R. Burnashev, 2023 International Russian Smart Industry Conference (SmartIndustryCon), 690-694 (2023)
13. R. Burnashev, A. Enikeeva, I.F. Amer, A. Akhmedova, M. Bolsunovskaya, A. Enikeev, Lecture Notes in Networks and Systems 544 (2023)