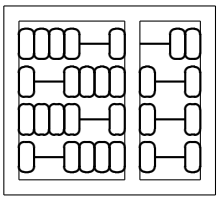
UNIVERSITY OF CAMPINAS

INSTITUTE OF COMPUTING



Dwsr: Predicting the memory usage of tensorial algorithms

Student: Daniel De Lucca Fonseca

Advisor: Prof. Edson Borin

Abstract: Lorem ipsum dolor

Contents

1 Introduction

Sit amet lorem

2 Background

Sit lorem dolor

3 Related Work

Lorem ipsum

4 Research proposal

This work aims to develop a new approach towards estimating the maximum required memory to compute a graph. Based on the current state-of-the-art, this task can be resolved with the following strategies:

- **Contextual analysis**: Estimates the maximum required memory based on information unrelated to the graph itself. This is feasible and useful for schedulers, since they can correlate user-behavior with resource usage. However, this approach is not suitable for general-purpose memory estimators, since it does not take into account the graph itself;
- **Historical analysis**: Can estimate the maximum required memory of a graph by training a model using historical data. This approach is suitable for graph-specific memory estimators, but not suitable for general-purpose usage, since it requires a large amount of historical data to be collected and processed.

Both approaches are suitable for some specific scenarios, but not for general-purpose usage. The proposed approach aims to overcome the current limitations by implementing: (i) a feature extractor capable of gathering information from both the source code and the input data and (ii) a reinforcement learning model capable of being used without the need for labeled historical data.

This section is organized as follows: first on subsection 4.1 presents details regarding a practical use-case related to this research proposal; subsection 4.2 defines the problem, exploring open issues that I expect to explore; subsection 4.3 states questions and hypothesis that I am planning to

experiments towards during the research; subsection 4.4 gives more details towards the experiments I am planning to execute; finally, subsection 4.5 explains, in detail, the deliverables and steps of the research itself.

4.1 Use case: Seismic data processing

A sample use-case for the proposed research is processing seismic data. On such graphs, a common issue is understanding if the current implementation of the graph is able to process the input data. This is a common problem, since the input data is usually very large. To address this issue, the graph is usually implemented in a distributed fashion, where each node is responsible for processing a small portion of the data. In such cases, the user usually tries to achieve the data distribution optimal point. Since adding more nodes to the computation increases network overhead, the user tries to find the optimal number of nodes to achieve the best performance.

4.1.1 DASF

DASF [1] is a graph processing framework that is capable of processing seismic data. The library is composed of a set of seismic attributes that can be composed into a graph and executed against a seismic dataset. The graph is executed in a distributed fashion, where each node is responsible for processing a small portion of the data. The distributed execution is configurable by the user, who can choose the *batch size* during the execution.

As mentioned before, choosing the right batch size is a critical step for the execution, since setting a batch size that is too small can lead to a high network overhead, while setting a too large batch size can lead to errors due to the lack of memory.

Based on this, a critical step of the research is to develop a tool that is capable of being integrated as a plugin into DASF [1]. This will be covered in more detail on subsection 4.4.

4.1.2 Seismic operators

During the research I aim to experiment using many different seismic operators. Each operator is designed to execute a specific task, usually by computing a specific attribute. A common characteristic of these operators is that they usually don't have too many control parameters, which makes them a good candidate for the proposed research. As I am going to explain on subsection 4.2, a critical limitation for predicting the maximum required memory is the amount of control statements. Based on this, experimenting with operators that have few control statements is a good

way to test the proposed approach.

4.2 Problems to be addressed

On this section I am going to present the problems that I expect to address during the research. Each problem is presented as a subsection with a brief description and a list of open questions that I expect to answer during the research. The compiled list containing the main research questions is presented on subsection 4.3.

4.2.1 Control statements

Assuming that the research is capable of finding a suitable way of extracting features from the source code, control statements could still potentially change the amount of memory required to execute the graph. During the research, we refer to algorithms with a low amount of control statements as *tensorial algorithms*. Based on this assumption, this research is going to focus on graphs containing only tensorial algorithms. As mentioned in subsection 4.1.2, seismic operators are a good candidate for this research, since they usually don't have too many control statements.

4.2.2 Language-agnosticy

Ideally, a general-purpose memory estimator should be able to work with any programming language. However, each programming language has its own peculiarities, which would require a specific feature extraction process for each language. For the purpose of this research, I am going to focus on graphs implemented in Python, since this is the language used by DASF [1].

4.2.3 Feature extraction

Based on what was presented on subsection 4.2.2, the feature extraction process is going to be language-specific. This is a critical step for the research, since this process is going to extract features from both the input and the source code. Although the feature extraction process is going to be language-specific, I aim to make the features extracted from the source code to be language-agnostic. If this is possible, the feature extraction process can be reused for other languages with minor changes.

4.2.4 Garbage collection

During the experiments a critical issue that I am going to face is the garbage collection. The garbage collection strategy may vary depending on the programming language. Python uses a reference counting strategy, where the garbage collector is responsible for freeing the memory of objects that are not being used anymore. Although this strategy is very efficient, it can mimick the real amount of memory required to execute the graph if the machine still has enough memory to execute the graph.

It is also important to mention that the garbage collection strategy also varies depending on the processing strategy. Python algorithms that are being executed by the CPU are going to use the default Python garbage collection strategy, while algorithms that are being executed by the GPU are going to use a different strategy.

4.2.5 Work memory

During runtime, an algorithm may require a temporary memory space to store intermediate results. This is a common strategy, widely used by many algorithms. The amount of working memory in a specific point in time usually depends on both those intermediate results, as well as the data being handled by the algorithm. Although those intermediate results can easily be identified as variables, the value being held into those variables are not easy to identify.

4.2.6 Data transformation

From the perspective of the graph itself, most algorithms are going to store some value after the execution. The strategy may vary, while some may mutate the input data, others may return a new data structure. Since the research aims to explore the memory usage of the graph itself, this is a critical issue that needs to be addressed.

4.2.7 Real-time usage metrics collection

During the execution of the model, many usage metrics related to the memory usage are going to be collected. Gathering those metrics, as well as integrating those within the reinforcement learning process is a critical step for the research. To improve efficiency, the model should be able to predict on partial results, removing the need to improve itself only after the execution is finished. Based on that, gathering real-time usage metrics may be a good way to improve the efficiency of the model.

4.3 Research questions

Based on everything that was presented so far, this section contains a list of open questions that are going to be addressed during the research. Those questions are going to be used as guides for the experiments that are going to be executed.

- **RQ1**: What is the maximum rate of control statements that still gives a good memory usage prediction?
- **RQ2**: Which are the most important features to be extracted from the source code?
- **RQ3**: Which are the most important features to be extracted from the input data?
- **RQ4**: How to gather the real amount of memory being used considering Python's garbage collection strategy?
- **RQ5**: What is the proper way of handling CPU vs. GPU processing when estimating the memory usage?
- **RQ6**: What are the features that are capable of identifying intermediate results being held during the execution of an algorithm?
- **RQ7**: How to identify the data transformation pattern within a graph?
- **RQ8**: How to gather metrics during the execution of a graph?
- **RQ9**: How to integrate the gathered metrics within a reinforcement learning model?

4.4 Research methodology

4.5 Work plan and schedule

References

[1] J. Faracco, O. Napoli, and E. Borin, *Dasf is an accelerated and scalable framework*, `https://github.com/discovery-unicamp/dasf-core`, 2023.