Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

Grau en Enginyeria Informàtica (GEI)

Computació

Development of a Tensor algebra DSL for HPC platforms

Roger Vilaseca Darné
(`roger.vilaseca.darne@estudiantat.upc.edu`) // (`roger.vilaseca@bsc.es`)

Director:
Dr. Eduard Ayguadé Parra, Computer Architecture Department

Codirector:
Dr. Vicenç Beltran Querol

Barcelona, 26th October 2020

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Abstract

In recent years it has been created a new model to generate domain-specific languages called Multi-Level IR, this model lets to optimize the different depictions of the code. To try this new model has been decided to develop a domain-specific language about the tensor algebra. This project will let the creation of a new DSL that will facilitate the work of many experts when it is needed to make calculations of this type of data structure.

Resum

En els darrers anys s'ha creat un nou model per generar llenguatges de domini específic anomenat Multi-Level IR, aquest model permet optimitzar les diferents representacions del codi. Per tal de provar aquest nou model, s'ha decidit desenvolupar un llenguatge de domini específic sobre l'àlgebra de tensors. Aquest projecte permetrà engegar la creació d'un nou DSL que facilitarà la feina de molts experts a l'hora de realitzar càlculs amb aquest tipus d'estructures de dades.

Resumen

En los últimos años se ha creado un nuevo modelo para generar lenguajes de dominio especifico llamado Multi-Level IR, este modelo permite optimizar las diferentes representaciones del codigo. Por tal de probar este nuevo modelo, se ha decidido desarrolar un lenguage de dominio especifico sobre la algebra de tensores. Este proyecto permitirá empezar la creacion de un nuevo DSL que facilitará el trabajo de muchos expertos en el momento de realizar calculos con este tipo de estructura de datos.

Table of Contents

List of Figures

List of Tables

1 | State of the art

Domain-Specific Languages (DSLs) [1] are small programming languages designed to be used in the solution of problems from a specific domain.

Some times these DSLs need different levels of abstraction to optimize the generated code, here appears the Multi-Level Intermediate Representation (MLIR) [2, 3].

The project will be focused on the creation of a Tensor Algebra DSL using the MLIR tool, taking advantage of all its potential.

1.1 Context

This project is a Bachelor Thesis from the Bachelor Degree in Informatics Engineering done by the Facultat d'Informàtica de Barcelona (FIB) being part of the Universitat Politècnica de Catalunya - BarcelonaTech (UPC). This project has been done in the Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS) with the purpose to research in the creation of a DSL through MLIR and see the potential of this IR.

MLIR has been designed to let multiple support requirements in a unified infrastructure. But we will focus on the following:

- Representation of kernels for ML operations in a form suitable for optimization.
- Ability to host high-performance-computing-style loop optimizations across kernels (fusion, loop interchange, tiling, etc) and to transform memory layouts of data.
- Code generation "lowering" transformations such as DMA insertion, explicit cache management, memory tiling, and vectorization for 1D and 2D register architectures.

To do it, we will reproduce a subset of The Tensor Algebra Compiler (TACO), which is a C++ library created to compute tensor algebra expressions on sparse and dense tensors [4, 5].

1.2 Problem formulation

MLIR has been designed to be a new tool in the creation of DSLs, but being such a new tool, nobody knows what its maximum potential is. Still, we know that it can be a great tool to support multiple different requirements.

TACO is an existing library created to compute tensor algebra expressions. This library has already defined a domain and has an open repository, which will be very useful in the task of implementing a subset of it using MLIR.

A TACO DSL will be created, which will have the name of TEnsor ALgebra DSL (TEAL DSL). This DSL will generate kernels that will solve all kind of operations with even sparse or dense tensors in the future.

Even though in this project will be developed only the dense tensors and the following operations.

- Addition and subtraction
- Tensor-vector multiplication

When the project starts, our ambition wanted to create the whole DSL during the planned months. But when we begin to develop it, we realized that this would be unreachable, and finally decided to reduce it.

1.3 Stakeholders

- **Companies** - There are lots of companies nowadays that work with large groups of data, generally represented in sparse form. A DSL that solves problems with this data efficiently will let the researchers in that company work with them without knowing how doing the complex computation of them.
- **Researcher** - The researcher will be the responsibility to achieve the project, that includes doing the planning, development, implementation and documentation. Also will have to make conclusions and an auto review about the project.
- **Project director** - The project director will be the person who guides the researcher to achieve the objective. He will do that doing weekly meetings with the researcher to see the evolution of the project.

2 | Motivation

MLIR is a technology still in development that was presented in February 2019 by the Tensorflow group on Google, and since the 24th of December 2019, was developed by them. After that day the project was moved to the LLVM project, who are now the leading developers of MLIR.

For this reason, there are not many examples done with MLIR yet, but the BSC committed with this project and want to know how powerful it is.

In that point, we had to find which is the best project to implement with MLIR. First of all, we thought about Saiph; a DSL developed at BSC for simulating physical phenomena modelled by Partial Differential Equations systems [6]. The problem with this DSL is that it is still in development and this makes it difficult for us to create our version of the DSL with MLIR.

After a few more research we came across with TACO, a C++ library used as a Tensor Algebra Compiler. We found this library interesting because it was well delimited and defined, also has his repository on GitHub [5].

Finally comparing both options, we decided in favour of creating a new DSL based on the TACO library, leading to the final user of the DSL the opportunity to do tensor algebra operations without the necessity of knowing any previous programming language.

3 | Project Scope

It is impossible to start a project without defining the objectives and how to reach them. Also, we have to take into account the problems that can appear during the process of development.

3.1 Project objectives

The main objective of this project is to research with the MLIR tool and to know its potential. To do this, it will be compared the ease to create a DSL using MLIR with the use of other similar tools; calculate the execution times of the created DSL, trying to minimize them as much as possible.

From the last objective, the objective of creating a tensor algebra DSL appears. Based on this significant objective, we will have to delimit the subset of operations to implement.

3.2 Requirements

3.2.1 Functional requirements

- The DSL will let the user enter a code based in a combination of operations based on Tensor Algebra.
- The DSL will return an LLVM code to be executed on the computer.

3.2.2 Non-functional requirements

- The code will be optimized to create a more efficient program.
- The DSL will return errors and warnings explaining to them if the code provided is not correct.

3.3 Potential obstacles and Risks

3.3.1 MLIR in development

In some point of the development of the DSL, we could be reached a point where we could get stuck without knowing how to continue. This problem comes with a huge obstacle, the lack of documentation of MLIR because it is a very new tool and there are some things still in development.

To overcome this obstacle, we will have to use the MLIR Google Group [7]. The use of this group will be beneficial but sometimes to get a question may pass too much time.

3.3.2 Virus / fatal error on laptop

In any computer environment, can appear different problems like virus, memory corruption, an unexpected closure etc. This problem may finish with the loose of previous developing hours or in an extreme case can cause irreparable damage to the laptop.

In order to minimize the problems occasioned by this risk, we have to save in the cloud our project continuously, avoiding losing as much information as possible. To do it, we will use GitLab [8] to keep our project in the cloud, even though the problem will cause a loss of hours to repair the laptop and prepare it for use.

3.3.3 Other kind of virus

It can appear other kinds of virus in our society that creates difficulties to do group projects.

If this happens, we will adapt to work in a decentralized manner and use the existing technologies of virtual conferences to work with the group.

4 | Methodology

For the project, we will work using an Agile methodology, which lets us have a stable and feasible development.

4.1 Working method

The project will be done using an iterative method, based on having stable versions of the project and working with the newest version to create the next one. Each iteration will be divided into the following parts:

1. Analyze the status of the project
2. Set the improvements to be done in this iteration
3. Develop the improvements
4. Evaluate the result

The iterations will be done until reaching the goal of the project.

We will create some test suites to check the correctness of the completed iteration; it will show the correctness of all the program.

4.2 Tracking tools

In order to implement this methodology, it will be used GitLab [8], an Open Source tool used to develop collaborative software and to handle version control.

In this repository, it will be able the stable versions of the project, which the director and people of the working group have access. There is also a private part where only I have access where it will be the source code of the iteration that is being created at that moment.

4.3 Validation method

In order to maintain the project correctly focused, the working group will do weekly meetings. In which it will be evaluated the work done during the previous week and to discuss, which will be the next iterations to develop.

5 | Time planning

This chapter defines the planning of the project. The project is split into different tasks, which has dependencies between them and needs some resources. This will be illustrated with a Gantt diagram in section 5.6.

In this part, it will be discussed the setbacks that can appear, how could affect the plan and the resources, and how it can be fixed.

The time planning has been modified because before the project starts, it seemed to us that the creation of the first part of the project would be more straight forward. After the first month of development, we realized that the parallelization and the design of the sparse tensors would be unreachable. For that reason, we modified the time planning to accurate the objective.

5.1 Task description

Here it will be explained the different tasks that compromise the project.

5.1.1 Project Management

This tasks regards to the subject *Gestió de Projectes* (GEP). This task is distributed in smaller once regarding the different deliverables.

- **ICT tools**: 4 hours
- **Context and Scope**: 24.5 hours
- **Time planning**: 8.25 hours
- **Budget and sustainability**: 9.25 hours
- **Personal and professional skills for Project and Team Management**: 2.25 hours
- **Final document**: 18.25 hours

During the course of the subject it will be used: LaTeX, *Visual Studio Code* as the editor, *pdflatex* as the compiler, *Google*, *Atenea* and *El Racó*.

5.1.2 Learning MLIR

As it is explained in this section 3.1 to create the DSL, it will be used MLIR. As MLIR it is one of the newest created IRs, it will need a certain amount of time to learn how MLIR works. In this task, it will be needed to analyze the code in order to use each functionality properly. It will be used the *Toy* example, created to learn the different parts that have to be followed to create a DSL. Finally, to practice before starting our DSL, it will be implemented new functionalities to the *Toy* example.

The time for this task it will be distributed in the following way:

- **What's MLIR?**: 40 hours
- **Study the Toy Example**: 120 hours (About 20 hours for chapter)
- **Implementing new features**: 60 hours

It will be used: *Git*, *Gitlab*, *Github*, *Visual Studio Code*, *Vim*, *GCC* and *Bash.*

5.1.3 Learning Tensor Algebra

Tensor Algebra is an essential part of the project, such it is the theme of the DSL. This creates a necessity of studying the domain.

For this part, it will be used: *Google*, *Youtube*, books, and articles related to this topic.

5.1.4 TEAL DSL description

One of the essential parts of the creation of the DSL is to determine which will be the syntax of the DSL and which functionalities will have the DSL. If this part is not done properly, it will cause lots of problems in the process of implementation.

For this part, it will be used any available code editor to define it.

5.1.5 Develop of the TEAL Language

In this part, we will generate the lexer, the parser and the resulting Abstract syntax tree (AST) of our language. This part is essential to do it properly in the begging because if there exists an error or an inconsistency in this part, we might drag it to the next tasks.

C++ will be used as the implementation language, using the *GCC (GNU Compiler Collection)* in order to compile it and *Visual Studio Code* as code editor.

5.1.6 Lowering to the TEAL Dialect

After getting the ASTwe will create the TEAL Dialect, which consists of the different kind of operations that will be able to resolve the TEAL DSL. This task will be done in parallel with the lowering to the MLIR dialects because each operation will be developed to the end.

C++ will be used as the implementation language, using the *GCC (GNU Compiler Collection)* in order to compile it and *Visual Studio Code* as code editor.

5.1.7 Lowering to MLIR Dialects

MLIR has a set of dialects already created, that lets the programmers transform their dialects to these and then make the lowering to LLVM easier. In our project, we will use the Affine Dialect, the Standard Dialect and the SCF Dialect. These dialects will let us create the loops, work with the part of the tensor that we need, do basic operations with the tensors etc. Firstly it will be done a Lowering to Affine and Standard Dialects and then to SCF and Standard Dialects.

It will be used the same resources than in Section 5.1.6.

5.1.8 Lowering to LLVM

The final implementation will consist of converting the resulting code of the previous task to a language that any computer could understand, in this case, LLVM.

It will be used the same resources than in Section 5.1.6.

5.1.9 Evaluation

In this part, the DSL will be tested to evaluate the correctness of our DSL, generating tests and checking if it returns the desired results.

C++ will be used as the implementation language to create the problems, using the *GCC (GNU Compiler Collection)* in order to compile it and *Visual Studio Code* as code editor.

5.1.10 Thesis writing and defence

All the research done will be concentrated in the thesis document with all the knowledge and results obtained. Afterwards, a slide presentation has to be done to defend it.

It will be used: LaTeX, *Visual Studio Code* as the editor and *pdflatex* as the compiler in order to do the documentation and *Microsoft PowerPoint* in order to do the slides for the defense.

5.2 Timing

It will be assigned an approximate duration to each task specified in Table 5.1. The duration are explained in 5.1.

Task	**Time (h)**
Project Management	66.5
Learning MLIR	220
Learning Tensor Algebra	40
TEAL DSL description	40
Develop of the TEAL Language	50
Lowering to the TEAL Dialect	50
Lowering to MLIR Dialects	175
Lowering to LLVM	25
Evaluation	33.5
Thesis writing and defense	60
Total	**760**

Table 5.1: Estimated time in hours for task.

5.3 Task dependencies

In order to do some of the tasks described in 5.1, other tasks must be completed. The prerequisites are exaplained here in Table 5.2.

5.4 Resources

To do this project, it will be needed several resources.

5.4.1 Hardware

- **Laptop**: DELL Latitude 7490
  - CPU: Intel® Core™ i7-8650U
  - Memory: 16GB
- **High Performance Computer**: We already don't have it.

Task	Prerequisites
Project Management	-
Learning MLIR	-
Learning Tensor Algebra	-
TEAL DSL description	Learning MLIR Learning Tensor Algebra
Develop of the TEAL Language	DSL description
Lowering to the TEAL Dialect	Develop of the TEAL Language
Lowering to MLIR Dialects	Lowering to the TEAL Dialect
Lowering to LLVM	Lowering to MLIR Dialects
Evaluation	Lowering to LLVM
Thesis writing and defense	Evaluation (But some documentation can start previously)

Table 5.2: Prerequisites for task.

5.4.2 Software

- **Ubuntu Linux**: GNU/Linux distribution on the laptop.
- **Git**: Decentralized version control system.
- **GitLab**: Git repository management server, used by the BSC.
- **GitHub**: Git repository management server, used by hosting MLIR.
- **C++**: Main programming languaged used to develop the project.
- **GCC**: GNU compiler, used to compile the MLIR code.
- **GDB**: GNU debugger, used to fing bugs in the C++ codes.
- **Visual Studio Code**: Text editor.
- **LaTeX**: Computer typewriting system.
- **Microsoft Office 365**: Online office package used to create slides.
- **Google Chrome**: Web browser.

5.4.3 Human Resources

In this project will participate, two directors, some support staff and the developer, which is the student.

5.4.4 Spaces

The spaces for this project will be in the BSC:

- A desk where the project will be developed.
- A meeting room where will be done the weekly meetings with the project director.

5.5 Workarounds and action plan

It can be found different types of workarounds, that can cause a different type of deviations of the project plan.

If the workaround found is a small one, like a bug in the development part, as the methodology used is an Agile one, the sprints that appear weekly may be extended causing the necessity of doing a little more hours in the problem.

If it is bigger, the workaround can cause an extension at the end of the project. This is possible given that the project is set to end in August, but it should not be delivered and defended until October.

5.6 Gantt diagram

The Gantt diagram is divided into two figures, the diagram of the 2019 and the diagram of 2020. Each Gantt diagram shows in the ”y” axis month and week number. Each week in the Gantt diagram will represent an amount of work of 20 hours. When there are tasks overlapping, the 20 hours will be distributed between them.

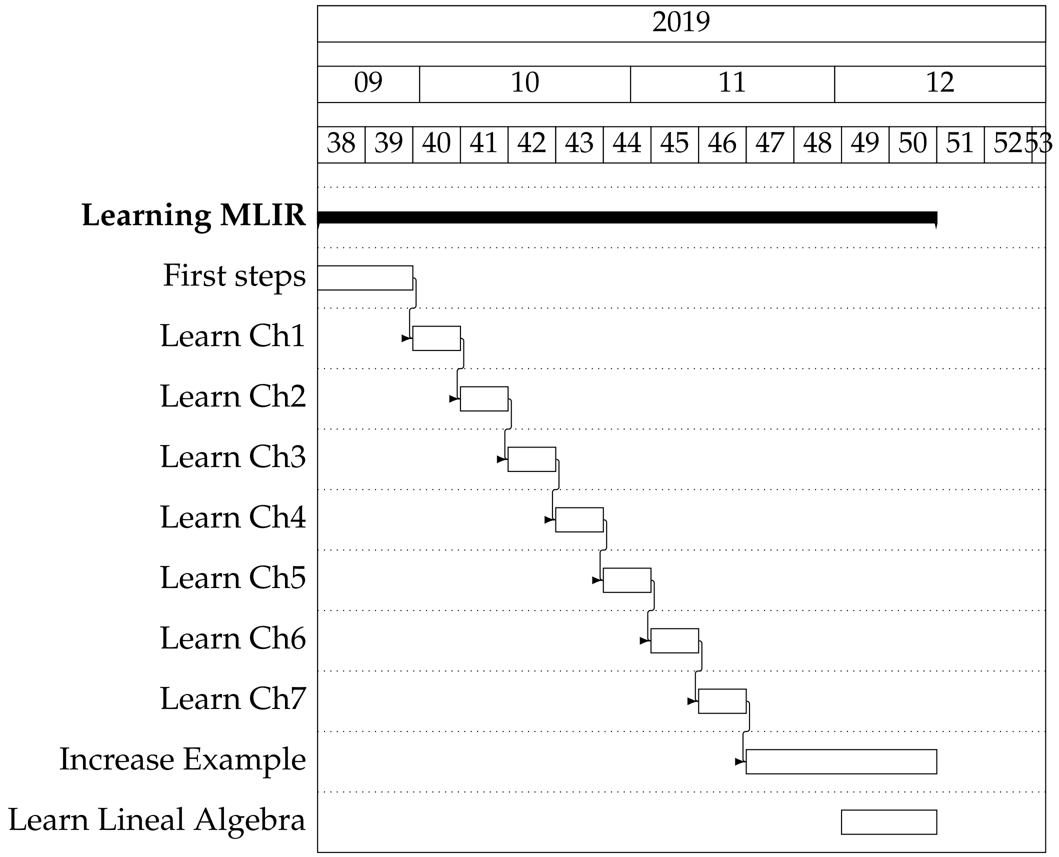

Fig. 5.1: Gantt diagram of the project 2019

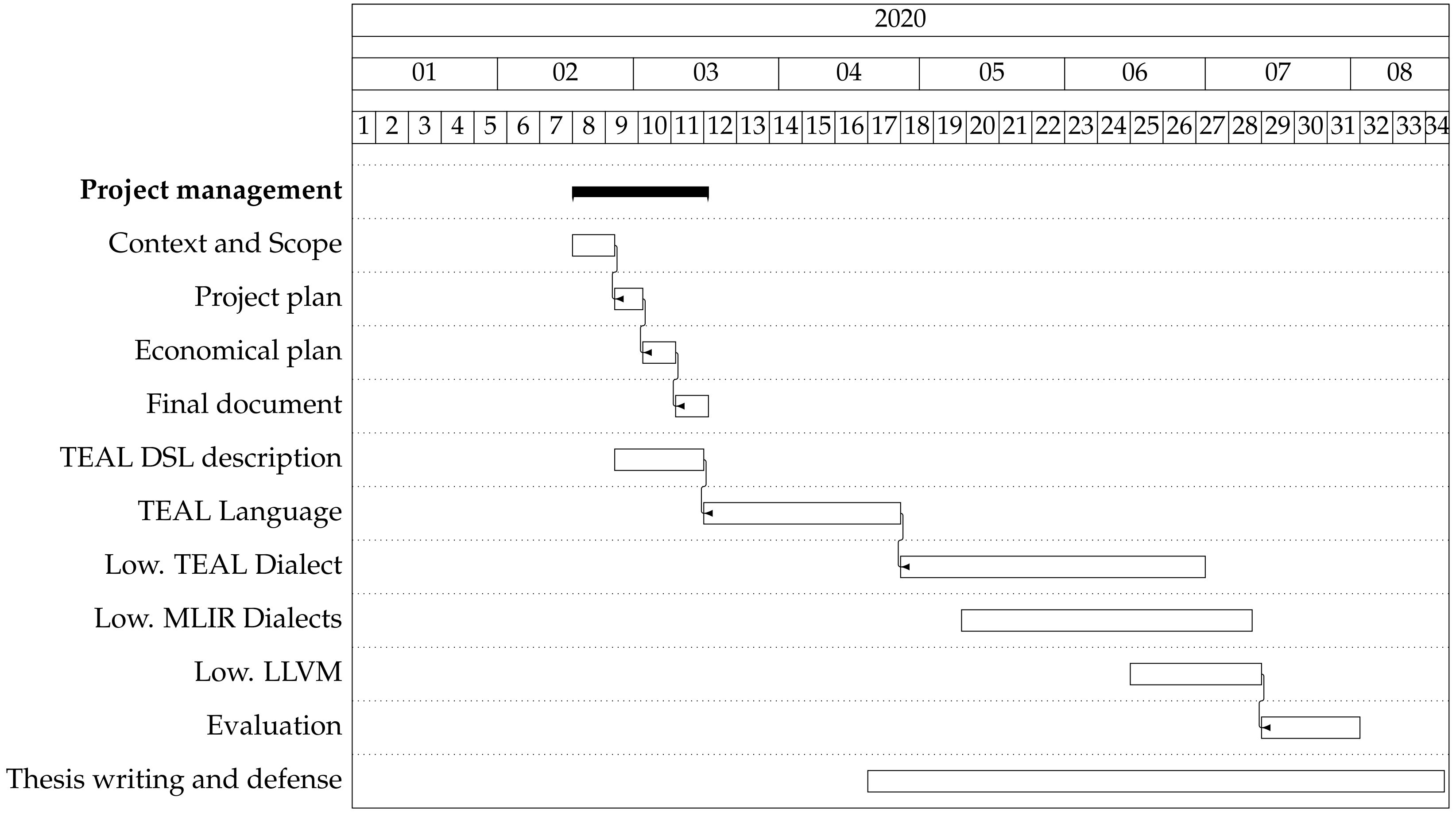

Fig. 5.2: Gantt diagram of the project 2020

6 | Economic Management

This chapter shows the planned budget for the project. There is also the possible deviations in the budget.

6.1 Budget

6.1.1 Direct costs

The direct costs for the project are those that are derived directly from the project.

Human resources

In the project, there are different types of people working on it. First of all, we take into account the Informatic Engineering student who is developing the software, paid by the student cooperation contract for the UPC. There will also be other BSC Staff like the director who spends one hour per week in the project tracking and some workmates that give some help in problems that they are experts.

Resource	**Price (€/h)**	**Amount (h)**	**Total (€)**
Informatic Engineering Student	9	760	6840
Project Director	20	43	860
Experts	20	60	1200
Total			8900

Table 6.1: Cost of human resources.

All the costs calculated above are gross salary, this means that the national insurance and IRPF are included.

Software resources

The price of the software will be calculated based on the cost of each product, how many time can be used and, the amortization of it.

Resource	Price (€)	Life (months)	Amortization (€/h)
Ubuntu Linux	0	-	0
Git, GitLab, Github	0	-	0
GDB, GCC, C++	0	-	0
Vim, VSCode	0	-	0
LaTeX	0	-	0
Microsoft Office 365	7 /month	2	0.7
Google Chrome	0	-	0
Total			0.7

Table 6.2: Cost of software resources.

As shown in Table 6.2, most of the software is free or Open Source, which means that it has no cost. In the final budget will only appear the ones that have to be paid.

Hardware resources

The costs regarded to the Hardware are explained in Table 6.3. The amortization will be calculated, taking into account that the machines will be used after the project.

Resource	Price (€)	Life (years)	Use (hours/day)	Amortization (€/h)
BSC Laptop	2000	4	4	0.46
Total				0.46

Table 6.3: Cost of hardware resources.

6.1.2 Indirect costs

Indirect costs are those caused by the project but not directly.

It is taken into account the power of the laptop and the high-performance computer, a flat internet rate and the costs of an office in Barcelona.

Resource	Price	Amount	Total (€)
BSC Laptop Power	0.1349 €/KWh	65 W * 760 h	6.67
Internet	40 €/month	12 months	480
Office	260 €/month	12 months	3120
Furniture	-	-	1500
Office services	125 €/month	12 months	1500
Total			6606.67

Table 6.4: Indirect costs.

6.2 Risk management

There is one primary risk contingency in our project; thus the number of hours have been calculated counting the hours that the developer might spend fixing bugs, this contingency will be global and based on a 10%. This quantity will be enough to replace any material would need to be bought in any setback scenario, and the time related to its installation.

6.3 Final budget

Taking into account all the costs described in Tables 6.1, 6.2, 6.3 and 6.4, the Table 6.5 sum ups the budget of the project.

Resource	Units	Price (€/unit)	Total (€)
Project management			
BSC Laptop	66.5h	0.46	30.59
Student	66.5h	9	598.5
Learning MLIR			
BSC Laptop	220h	0.46	101.2
Student	220h	9	1980
Project Director	13h	20	260
Expert	20h	20	400
Learning Tensor Algebra			
BSC Laptop	40h	0.46	1.84
Student	40h	9	360
Project Director	2h	20	40
Expert	2h	20	40
TEAL DSL description			
BSC Laptop	40h	0.46	1.84
Student	40h	9	360
Project Director	3h	20	60
Expert	2h	20	40
Develop of the TEAL Language			
BSC Laptop	50h	0.46	23
Student	50h	9	450
Project Director	3h	20	60
Expert	5h	20	100
Lowering to the TEAL Dialect			
BSC Laptop	50h	0.46	23
Student	50h	9	450
Project Director	3h	20	60
Expert	6h	20	120
Lowering to MLIR Dialects			
BSC Laptop	175h	0.46	80.5
Student	175h	9	1575
Project Director	11h	20	220

Expert	20h	20	400
Lowering to LLVM			
BSC Laptop	25h	0.46	11.5
Student	25h	9	225
Project Director	2h	20	40
Expert	3h	20	60
Evaluation			
BSC Laptop	33.5h	0.46	15.41
Student	33.5h	9	301.5
Project Director	3h	20	60
Expert	2h	20	40
Thesis writing and defense			
BSC Laptop	60h	0.46	27.6
Microsoft Office 365	20h	0.7	14
Student	60h	9	540
Project Director	4h	20	80
Indirect costs			
BSC Laptop Power	760h	0.0087	6.67
Internet	12 months	40	480
Office	12 months	260	3120
Furniture	-	-	1500
Office services	12 months	125	1500
Subtotal			15837.15
Contingency	10%		1583.72
Total without VAT			17420.87
VAT	21%		3658.38
Total			21079.26

Table 6.5: Final budget.

7 | Sustainability

7.1 Self-assessment of the current domain of sustainability

By completing the questionnaire, I have realised my degree of knowledge regarding the sustainability of the projects associated with ICT. At first, I realised that the vast majority of aspects to be valued had a specific experience, but not the result of which I have learned to the subjects of the career, but in courses that I have done outside the field of ICT, but that knowledge can be brought to them.

In the economic aspect, I have realised that there are many aspects to take into account when carrying out an ICT project, but many hits, at first sight, are not taken into account. This can end up causing a lack of fluid when carrying them out. Even so, I believe that having met them during the creation process, all possible variables have been included.

In the social aspect, I have always believed that when a project has to be developed, there must be moral grounds, it might not have groups of society that feel harmed by the created project. Even so, I think that you can always learn more about this topic and therefore be able to deepen and take it into account in future projects.

Finally, in the environmental aspect, I believe that our western society currently knows, but does not take into account, the significant problems that ICTs cause in the world at the time of developing new projects. That is why I believe that a worldwide effort is needed, including myself, in minimising the environmental impact of the technologies that are generated.

7.2 Economic dimension

Reflection on the cost you have estimated for the completion of the project

First of all, when the cost has finished, it seemed to me that the price was too high. After reviewing the calculus, I realised that the cost was all correct. After thinking about it, I realised that the vast amount of budget was related to the indirect costs associated with the project, because the second significant amount was the destinated hours, but them where necessary. The cost of renting an office in Barcelona is very elevated even for the places destined in the creation of new projects.

How are currently solved economic issues related to the problem that you want to address? How will your solution improve economic problems concerning other existing solutions?

Nowadays, there is no solution for the Tensor Algebra that generates a program directly from an expression. This means that companies that have to work with this kind of problems will arrive at a solution faster than before answers.

7.3 Environmental dimension

Have you estimated the environmental impact of the project?

The software will be implemented in a machine specialised in the operations that will generate the DSL. This will cause a decline in consumption in comparison to the use of other devices.

Did you plan to minimise its impact, for example, by reusing resources?

Some of the resources of the project are reused; for example, the laptop used to work, have been used for previous researchers in the company. Also, the HPC machine will be used after the project is finished.

How is currently solved the problem that you want to address? How will your solution improve the environment concerning other existing solutions?

As it is explained previously, the project tries to parallelise the solution, generating an improvement in the speed during the execution.

7.4 Social dimension

What do you think you will achieve -in terms of personal growth- from doing this project?

The project is related to the topics I found most interesting during the degree. This means that improving with these topics will cause a better preparation for the jobs I would be able to do after finishing the degree.

How is currently solved the problem that you want to address? How will your solution improve the quality of life for other existing solutions?

Taking into account that there is not a real solution for this problem that generates an immediate answer, this project will let to the users to minimise the hours that they spend on resolving this type of problems.

Is there a real need for the project?

Nowadays, there is not an IT direct solution to the problem.

8 | Multi-Level IR

The Tensor Flow group in Google developed the Multi-Level IR (MLIR) [2] in 2019 and aimed to reduce the cost of building domain-specific compilers, connect existing compilers etc.

In early 2020 MLIR was integrated inside the LLVM project, making easier the tasks of improving the language and a better discussion place to extend it.

MLIR has been choosen to develop this project as it allows to perform optimizations at different levels of representation.

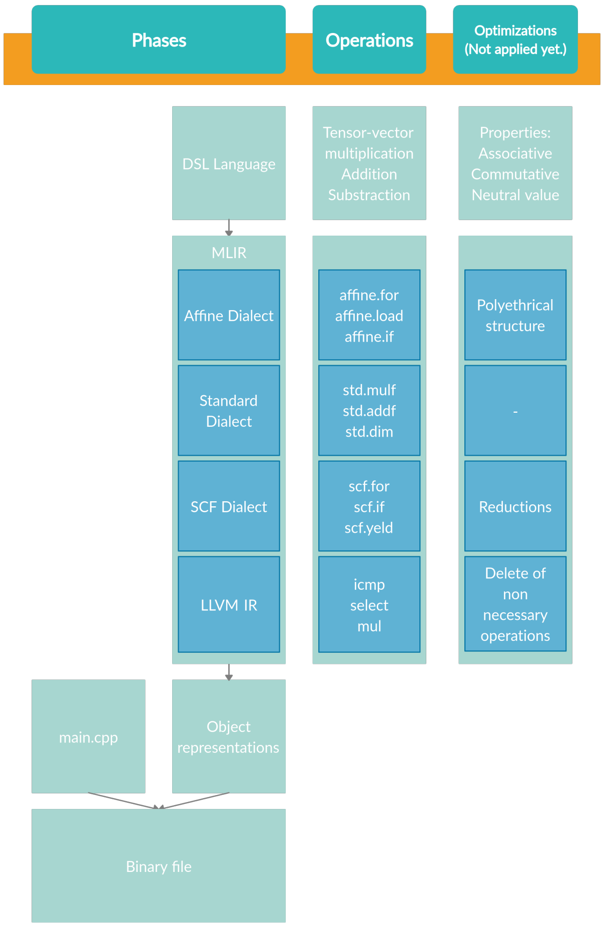
Phases
Operations
Optimizations (Not applied yet.)
DSL Language
Tensor-vector multiplication Addition Substraction
Properties: Associative Commutative Neutral value
MLIR
Affine Dialect
affine.for affine.load affine.if
Polyethrical structure
Standard Dialect
std.mulf std.addf std.dim
-
SCF Dialect
scf.for scf.if scf.yeld
Reductions
LLVM IR
icmp select mul
Delete of non necessary operations
main.cpp
Object representations
Binary file

Fig. 8.1: MLIR phases and optimizations

In this part of the project, it will be explained which resources of the MLIR will be used to generate our DSL.

8.1 Standard Types

The Standard Types ara the data structures used to represent a different kind of information that is needed to implement the DSL.

8.1.1 Integer

The Integer Type is a standard type used to represent integer numbers. There are different kind of sizes for Integer type going from one to thirty-two bits. It exists the possibility to use it only for natural numbers.

8.1.2 Floating-Point

The Floating Point Type is a type used to represent the real numbers. There are different kind of sizes for Floating-Point type going from sixteen to sixty-four bits.

8.1.3 MemRef

The MemRef Type is used to represent a reference to a region of memory. The MemRef allow the programmers to alloc memory and read and write data from the memory space.

MemRefs are descrived using dimensions, having each different dimension sizes, and having the possibility of having unknown sizes or an unknown number of dimensions. This makes it easier to interpret our tensors as MemRefs when are entered in functions when it isn’t known the sizes of the dimensions.

8.1.4 Index

The Index Type is used to represent natural numbers in order to use it to refer to the elements of the MemRef.

8.2 Dialects

Dialects are sets of defined operations, uses to separate different hardware and software targets.

MLIR contains eighteen Dialects, and the users use them to transform the languages that they create to equivalent languages formed by the operations that include. Also, MLIR has methods to convert the operations of one Dialect to another; in our project, this will be used to transform it until LLVM IR.

The following Dialects will be te used for the project:

8.2.1 Affine

The Affine Dialect contains operations used to creating loops and accessing the MemRefs Types. It is also specialized for Polyhedral Representations and Optimizations. The operations used are the following:

- *affine.for %i = %min to %max step %s*: Used to create loops. *%i* is an index that represents the induction variable, *%min* is the lower-bound value, *%max* is the upper-bound value and *%s* is the steps between each iteration.
- *%res = affine.load %A[%i]*: It is used to obtain a value stored in MemRef coordinates. *%A* is the MemRef from where we want to pick up the data, *%i* is a coordinate that we want to access (there can be more coordinates separated by commas if the MemRef has multiple dimensions) and *%res* is the value that has been loaded.
- *affine.store %val, %A[%i]*: It is used to save a value in MemRef coordinates. *%val* is the value that it is wanted to store, *%A* is the MemRef from where we want to pick up the data and *%i* is a coordinate that we want to access (there can be more indexes separated by commas if the MemRef has multiple dimensions).

8.2.2 STD

The Standard (STD) Dialect is used to create basic operations like additon or multiplication. The STD operations used in our program will be the following:

- *%a = std.addi %b, %c*: It is used to do an integer addition. *%a* is the result and *%b* and *%c* are the operands.
- *%a = std.addf %b, %c*: Similar to *std.addi* but with floats.
- *%a = std.muli %b, %c*: It is used to do an integer multiplication. *%a* is the result and *%b* and *%c* are the operands.
- *%a = std.mulf %b, %c*: Similar to *std.muli* but with floats.
- *%a = std.subi %b, %c*: It is used to do an integer substraction. *%a* is the result and *%b* and *%c* are the operands.

- *%a = std.subf %b, %c*: Similar to *std.subf* but with floats.
- *%a = std.diviu %b, %c*: It is used to get the quotient from an integer division. *%a* is the result and *%b* and *%c* are the operands.
- *%a = std.remiu %b, %c*: Similar to *std.diviu* but obtains the remainder.

There are other operations used from STD like *std.load*, but these aren't used directly by the programmer, these operations are obtained when doing a transformation from the Affine Dialect.

8.2.3 SCF

The final Dialect used in this project is the SCF Dialect, for generating loops or conditions. This Dialect isn't used directly by the programmer; the operations used are obtained in a transformation from the Affine Dialect.

9 | Tensor forms

In this part of the thesis, it will be explained how the Tensors will be considered for this project.

9.1 What's a Tensor

A Tensor is a data structure conformed by multiple elements; in this case, it will be formed only by doubles (8-byte floating-point number).

Tensors can have multiple dimensions having similarities with n-dimensional matrices.

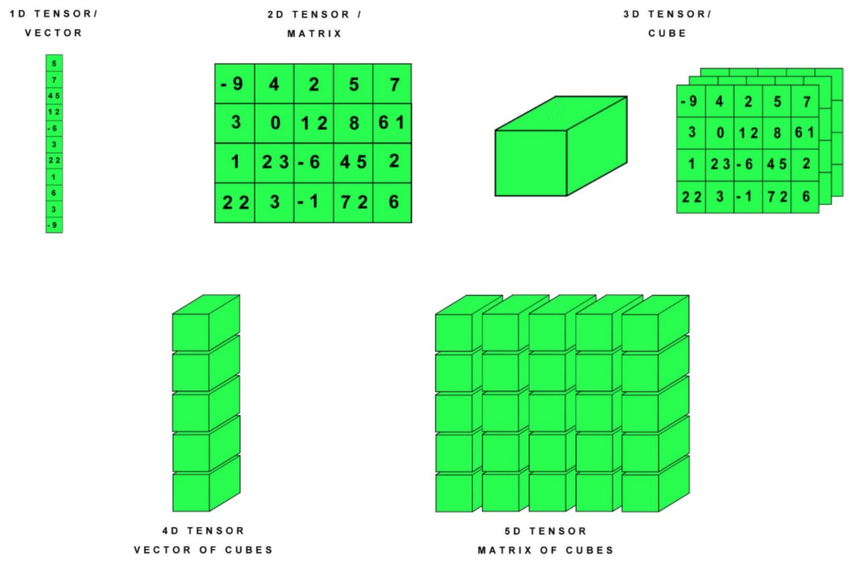

Fig. 9.1: Representation of a tensor

Each tensor will be formed by blocks and elements, making that for each dimension that we want for elements, it will be an extra one to create the blocks. For this project, we will only use four-dimensional tensors as matrices and two-dimensional tensors as vectors to delimit the scope of the project.

9.2 2 Dimensional Tensor (Vector)

Each tensor of this kind is represented using a MemRef of two dimensions, the first one with the number of blocks and the second one with the number of elements.

$$\begin{pmatrix} b0 \\ b1 \\ .. \\ bn \end{pmatrix}$$

Fig. 9.2: Blocks inside a Vector

$$\begin{pmatrix} e0 \\ e1 \\ .. \\ en \end{pmatrix}$$

Fig. 9.3: Elements inside a Block

$$\begin{pmatrix} b0/e0 \\ b0/e1 \\ .... \\ b0/en \\ b1/e0 \\ b1/e1 \\ .... \\ bn/en \end{pmatrix}$$

Fig. 9.4: Vector representation

9.3 4 Dimensional Tensor (Matrix)

The tensor that represents a Matrix follows a similar composition as in the vector, but this time with two dimensions for the elements and two for the blocks, as can be seen in the following figures.

$$\begin{pmatrix} b0,0 & b0,1 & ... & b0,n \\ b1,0 & b1,1 & ... & b1,n \\ ... & ... & ... & ... \\ bn,0 & bn,1 & ... & bn,n \end{pmatrix}$$

Fig. 9.5: Blocks inside a Matrix

$$\begin{pmatrix} e0,0 & e0,1 & ... & e0,n \\ e1,0 & e1,1 & ... & e1,n \\ ... & ... & ... & ... \\ en,0 & en,1 & ... & en,n \end{pmatrix}$$

Fig. 9.6: Elements inside a Block

$$\begin{pmatrix}
b0,0/e0,0 & b0,0/e0,1 & ... & b0,0/e0,n & b0,1/e0,0 & b0,1/e0,1 & ... & b0,n/e0,n \\
b0,0/e1,0 & b0,0/e1,1 & ... & b0,0/e1,n & b0,1/e1,0 & b0,1/e1,1 & ... & b0,n/e1,n \\
...... & ...... & ... & ...... & ...... & ... & ...... & \\
b0,0/en,0 & b0,0/en,1 & ... & b0,0/en,n & b0,1/en,0 & b0,1/en,1 & ... & b0,n/en,n \\
b1,0/e0,0 & b1,0/e0,1 & ... & b1,0/e0,n & b1,1/e0,0 & b1,1/e0,1 & ... & b1,n/e0,n \\
b1,0/e1,0 & b1,0/e1,1 & ... & b1,0/e1,n & b1,1/e1,0 & b1,1/e1,1 & ... & b1,n/e1,n \\
...... & ...... & ... & ...... & ...... & ... & ...... & \\
bn,0/en,0 & bn,0/en,1 & ... & bn,0/en,n & bn,1/en,0 & bn,1/en,1 & ... & bn,n/en,n
\end{pmatrix}$$

Fig. 9.7: Matrix representation

9.4 Operations Implemented

9.4.1 Addition and Substraction

This kind of operations are used when we have two tensors with the same number of dimensions, and the dimensions of each tensor also have the same size.

The operation consists in: for each element in each block it will be applied the addition (or subtraction) to the element in the same position in the other tensor, and then stored in the same place of a third tensor.

$$\begin{pmatrix}
0 & 1 & | & 2 & 3 \\
4 & 5 & | & 6 & 7 \\
-- & -- & -|- & -- & -- \\
8 & 9 & | & 10 & 11 \\
12 & 13 & | & 14 & 15
\end{pmatrix}
+
\begin{pmatrix}
0 & 1 & | & 2 & 3 \\
4 & 5 & | & 6 & 7 \\
-- & -- & -|- & -- & -- \\
8 & 9 & | & 10 & 11 \\
12 & 13 & | & 14 & 15
\end{pmatrix}
=
\begin{pmatrix}
0 & 2 & | & 4 & 6 \\
8 & 10 & | & 12 & 14 \\
-- & -- & -|- & -- & -- \\
16 & 18 & | & 20 & 22 \\
24 & 26 & | & 28 & 30
\end{pmatrix}$$

Fig. 9.8: Addition example

9.4.2 Tensor-Vector Multiplication (TVM)

This problem has made it easier only to perform the matrix-vector multiplication.

This operation consists in multiply each element of the first row (or column) for each element of the vector, add all the result and store it in the first position on the resulting vector, and repeat the process with the remaining rows.

$$\begin{pmatrix} 0 & 1 & | & 2 & 3 \\ 4 & 5 & | & 6 & 7 \\ -- & -- & -|- & -- & -- \\ 8 & 9 & | & 10 & 11 \\ 12 & 13 & | & 14 & 15 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ -- \\ 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 14 \\ 38 \\ -- \\ 62 \\ 86 \end{pmatrix}$$

Fig. 9.9: TVM by row example

10 | Implementing TEAL

In this part, it will be explained how each part of the TEAL DSL has been developed, the problems that have been appeared and how have been resolved.

As it can be seen in the following image, the process from lowering from the TEAL language to the LLVM IR consists in four transformations, one for each arrow, and five different representations, one for each square.

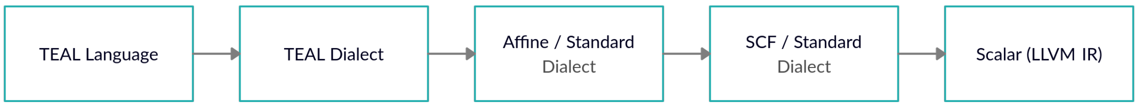

Fig. 10.1: Phases of the TEAL Compiler

10.1 TEAL Language

The first representation of TEAL DSL consists of the TEAL Language. The TEAL Language is the language that we have created to represent the operations that we want to use in this DSL. And the user will be in charge to know the language and provide it to the computer in a *".teal"* file.

```
y(i) = A(i,j) * x(j)
```

Fig. 10.2: Example of an operation in TEAL Language[1]

10.2 TEAL Dialect

In this part, we will convert the TEAL Language received in a set of operations that are known by the DSL.

[1]This example will be used in the different representations along with the explanation of each phase.

First of all we need to understand what has been given to us, creating a Lexer and a Parser that will generate an Abstract Syntax Tree (AST). The rules in order to understand the language are the following:

```
assign::= access '=' expr
expr::= term {('+' | '-') term}
term::= factor {('*' | '/') factor}
factor::= final | '(' expr ')' | '-' factor
final::= access | scalar
access::= identifier '(' varlist ')'
varlist::= var {',' var}
var::= identifier
```

Fig. 10.3: Parser rules[2]

A resulting AST will have this form:

```
Module:
    Assignment:
        Call 'y' (
        var: i
        )
        BinOp: *
        Call 'A' (
            var: i
            var: j
        )
        Call 'x' (
            var: j
        )
```

Fig. 10.4: Example of an AST from TEAL Language

After having the AST generated by the Parser, a checker analyze it, to see if the indexes provided are coherent (all the terms of addition has the same number of indexes etc.).

Finally, we go through the AST, converting each element in the desired part of the function. In our DSL each tensor will be interpreted as a MemRef Type 8.1.3.

[2]Some of the operations that Parser understands have not been developed, for example doing multiple operations, the minus or the division, but has been included for future work.

```
module {
    func @my_func(%arg0: memref<?x?xf64>, %arg1: memref<?x?x?x?xf64>, %arg2:
         memref<?x?xf64>) {
        "teal.tvm"(%arg0, %arg1, %arg2) : (memref<?x?xf64>, memref<?x?x?x?
           xf64>, memref<?x?xf64>) -> ()
        teal.return
    }
}
```

Fig. 10.5: Example of TEAL Dialect Representation

10.3 Affine/Standard Dialect

Once we have each operation described in TEAL Dialect, we proceed to do a lowering towards the Affine and Standard Dialects 8.2. The pass to this Dialects makes more comfortable the programmer work to arrive at the LLVM IR because Dialects are more Higher Level and the lowering to LLVM IR is straight forward.

For each operation that we obtained from the TEAL Dialect, we execute different functions that convert them to Affine and Standard Operations. In this part, we have to access the MemRefs taking into account the blocks and the elements.

After doing this lowering, we obtain an interpretation like Figure 10.6.

```
#map0 = affine_map<(d0, d1, d2, d3) -> (d0, d1, d2, d3)>
#map1 = affine_map<(d0, d1) -> (d0, d1)>
#map2 = affine_map<() -> (0)>
#map3 = affine_map<(d0) -> (d0)>


module {
    func @my_func(%arg0: memref<?x?xf64>, %arg1: memref<?x?x?x?xf64>, %arg2: memref<?x?xf64>) {
    %c0 = constant 0 : index
    %0 = dim %arg1, %c0 : memref<?x?x?x?xf64>
    affine.for %arg3 = 0 to #map3(%0) {
        %c1 = constant 1 : index
        %1 = dim %arg1, %c1 : memref<?x?x?x?xf64>
        affine.for %arg4 = 0 to #map3(%1) {
        %c0_0 = constant 0 : index
        %c2 = constant 2 : index
        %2 = dim %arg1, %c2 : memref<?x?x?x?xf64>
        affine.for %arg5 = 0 to #map3(%2) {
            %c3 = constant 3 : index
            %3 = dim %arg1, %c3 : memref<?x?x?x?xf64>
            affine.for %arg6 = 0 to #map3(%3) {
            %4 = affine.load %arg1[%arg3, %arg4, %arg5, %arg6] : memref<?x?x?x?xf64>
            %5 = affine.load %arg2[%arg3, %arg5] : memref<?x?xf64>
            %6 = mulf %4, %5 : f64
            %7 = affine.load %arg0[%arg4, %arg6] : memref<?x?xf64>
            %8 = addf %6, %7 : f64
            affine.store %8, %arg0[%arg4, %arg6] : memref<?x?xf64>
            }
        }
        }
    }
    return
    }
}
```

Fig. 10.6: Example of Affine and Standard Dialect Representation

10.4 SCF/Standard Dialect

After getting the representation using the Affine and Standard Dialects, we will do a Lowering to SCF and Standard Dialects, due to the lack of a direct lowering from Affine to LLVM IR. An example of this transformation can be found in Figure 10.7.

```
module {
    func @my_func(%arg0: memref<?x?xf64>, %arg1: memref<?x?x?x?xf64>, %arg2: memref<?x?xf64>) {
        %c0 = constant 0 : index
        %0 = dim %arg1, %c0 : memref<?x?x?x?xf64>
        %c0_0 = constant 0 : index
        %c1 = constant 1 : index
        scf.for %arg3 = %c0_0 to %0 step %c1 {
        %c1_1 = constant 1 : index
        %1 = dim %arg1, %c1_1 : memref<?x?x?x?xf64>
        %c0_2 = constant 0 : index
        %c1_3 = constant 1 : index
        scf.for %arg4 = %c0_2 to %1 step %c1_3 {
            %c0_4 = constant 0 : index
            %c2 = constant 2 : index
            %2 = dim %arg1, %c2 : memref<?x?x?x?xf64>
            %c0_5 = constant 0 : index
            %c1_6 = constant 1 : index
            scf.for %arg5 = %c0_5 to %2 step %c1_6 {
            %c3 = constant 3 : index
            %3 = dim %arg1, %c3 : memref<?x?x?x?xf64>
            %c0_7 = constant 0 : index
            %c1_8 = constant 1 : index
            scf.for %arg6 = %c0_7 to %3 step %c1_8 {
                %4 = load %arg1[%arg3, %arg4, %arg5, %arg6] : memref<?x?x?x?xf64>
                %5 = load %arg2[%arg3, %arg5] : memref<?x?xf64>
                %6 = mulf %4, %5 : f64
                %7 = load %arg0[%arg4, %arg6] : memref<?x?xf64>
                %8 = addf %6, %7 : f64
                store %8, %arg0[%arg4, %arg6] : memref<?x?xf64>
            }
            }
        }
        }
        return
    }
}
```

Fig. 10.7: Example of SCF and Standard Dialect Representation

10.5 Scalar (LLVM IR)

Finally, MLIR lets the programmer to do a direct lowering to the LLVM IR and creating an object file from the output. In Figure 10.8 can be found an example of the code.

```
70: ; preds = %67
    %71 = extractvalue { double*, double*, i64, [4 x i64], [4 x i64] } %43,
       1, !dbg !9
    %72 = extractvalue { double*, double*, i64, [4 x i64], [4 x i64] } %43,
       4, 0, !dbg !9
    %73 = mul i64 %53, %72, !dbg !9
    %74 = add i64 0, %73, !dbg !9
    %75 = extractvalue { double*, double*, i64, [4 x i64], [4 x i64] } %43,
       4, 1, !dbg !9
    %76 = mul i64 %58, %75, !dbg !9
    %77 = add i64 %74, %76, !dbg !9
    %78 = extractvalue { double*, double*, i64, [4 x i64], [4 x i64] } %43,
       4, 2, !dbg !9
    %79 = mul i64 %63, %78, !dbg !9
    %80 = add i64 %77, %79, !dbg !9
    %81 = mul i64 %68, 1, !dbg !9
    %82 = add i64 %80, %81, !dbg !9
    %83 = getelementptr double, double* %71, i64 %82, !dbg !9
    %84 = load double, double* %83, align 8, !dbg !9
    %85 = extractvalue { double*, double*, i64, [2 x i64], [2 x i64] } %50,
       1, !dbg !9
    %86 = extractvalue { double*, double*, i64, [2 x i64], [2 x i64] } %50,
       4, 0, !dbg !9
    %87 = mul i64 %53, %86, !dbg !9
    %88 = add i64 0, %87, !dbg !9
    %89 = mul i64 %63, 1, !dbg !9
    %90 = add i64 %88, %89, !dbg !9
    %91 = getelementptr double, double* %85, i64 %90, !dbg !9
    %92 = load double, double* %91, align 8, !dbg !9
    %93 = fmul double %84, %92, !dbg !9
    %94 = extractvalue { double*, double*, i64, [2 x i64], [2 x i64] } %32,
       1, !dbg !9
    %95 = extractvalue { double*, double*, i64, [2 x i64], [2 x i64] } %32,
       4, 0, !dbg !9
```

Fig. 10.8: Short example of LLVM IR Representation(3)

(3)The LLVM IR code generated has more lines that the showed example, to simplify the example only shows the first part of what is inside the innermost loop.

10.6 Reproduce the code

To reproduce the code, first of all, is needed to download an LLVM version from its GitHub project [9] (the appropriate commit is the cited here [10]).

Afterwards copy the files inside the source_code folder, from the delivered files in the additional material, into the *mlir/examples* folder from the downloaded LLVM folder.

Subsequently, copy what is inside the Examples folder into the *mlir/test/Examples* folder.

Finally, create a build folder in the source folder, enter, and execute it the following code (Using an Ubuntu OS):

```
cmake -G Ninja ../llvm \
-DLLVM_ENABLE_PROJECTS=mlir \
-DLLVM_BUILD_EXAMPLES=ON \
-DLLVM_TARGETS_TO_BUILD="X86;NVPTX;AMDGPU" \
-DCMAKE_BUILD_TYPE=Release \
-DLLVM_ENABLE_ASSERTIONS=ON

cmake --build . --target check-mlir

ninja tealc-v1
```

Once the code is compiled can be excuted with the following command:

```
./bin/tealc-v1 -emit=var ~/Documents/teal/mlir/test/Examples/Teal/Teal1/
    test1.teal
```

Where var can be:

- ast
- mlir
- mlir-affine
- mlir-scf
- llvm
- object

11 | Execution of the kernel

In this part of the thesis, it will be explained how to use the object file generated by the TEAL DSL to check the results. This will be done creating a C++ file that produces the Tensors (as MemRefs) and calls the function.

The object files contains a function with many MemRefs, which follows the following structure:

```cpp
template<typename T, size_t N>
struct MemRefDescriptor {
    T *allocated; //Pointer to the data
    T *aligned; //Pointer to the data if the element has a different alignment
    intptr_t offset; //Offset of the first element
    intptr_t sizes[N]; //The size of each dimension
    intptr_t strides[N]; //The number of elements to jump to the next index of each dimension
};
```

Fig. 11.1: MemRefs Data structure

Knowing how a MemRef it is formed, we can create arrays of doubles filled with the desired numbers.

After working a little bit with the MemRef, we found that the information is read in column-major order. In the following figure, it can be seen the order that follows in a Matrix (with blocks), if the original array contains the numbers from 0 to 15 ordered:

$$\begin{pmatrix} 0 & 2 & | & 8 & 10 \\ 1 & 3 & | & 9 & 11 \\ -- & -- & -|- & -- & -- \\ 4 & 6 & | & 12 & 14 \\ 5 & 7 & | & 13 & 15 \end{pmatrix}$$

Fig. 11.2: Column-major order

After the MemRefs are created the function created by the DSL has to be accessibly pointing out that comes from an extern file.

Finally, we only need to compile the created C++ file, linking it with the kernel in the object file and execute it.

In the files delivered in the additional material, there is a folder called kernel_executors, where you can find *".cpp"* files to reproduce it. Each file explains with which kernel has to be used.

12 | Conclusion

This project has been the starting point to create a new DSL for the BSC Company, creating a product that will make more accessible the tasks of Tensor Algebra computation.

At this point has been implemented a first version of the DSL, which lets the user do some basic operations related to the domain.

This project also gives to the student and to the company a brief example of the power that MLIR has, making much easier the task of programming DSLs.

Relatively with the budget, this hasn't been modified during the creation of the project. Many of the subprojects have been restructured, regarding the amount of time, but the sum of all of them is still the same.

When we started the project and we did not know much about the Tensor Algebra Domain, and about MLIR we thought that in the expected time it would be possible to develop more operations or more kind of tensors. Still, the lack of MLIR documentation and the considerable width of the domain forced us to simplify the objective.

Even though we think that starting this project will be essential for continuing working in the TEAL DSL and has been profitable for the creation of new ones using the MLIR tool.

13 | Future work

As it is explained in the Conclusion, this project has been only the start of a larger project.

Firstly, the most logical implementations would be:

- Add more operations to the DSL
- Add optimizations in operations
- Add new ways to read the tensors
- Add the possibility to recieve sparse tensors

Secondly, when the DSL is perfectly formed, it could be added new ways to lower to make it more efficiently in different machines where can be executed like TensorCore machines.

Finally, it could be interesting to add some parallelization in the generated kernels to make faster the execution of them.

Bibliography

[1] Markus Voelter and Sebastian Benz. *DSL engineering: designing, implementing and using domain-specific languages*. 2013. 1

[2] Multi-Level Intermediate Representation (MLIR), . URL https://mlir.llvm.org/. Accessed: (2020-10-18). 1, 24

[3] Chris Lattner and Jacques Pienaar. MLIR Primer: A Compiler Infrastructure for the End of Moore's Law, 2019. URL https://storage.googleapis.com. Accessed: (2020-02-19). 1

[4] The Tensor Algebra Compiler (TACO), . URL http://tensor-compiler.org/. Accessed: (2020-02-19). 1

[5] TACO Github, . URL https://github.com/tensor-compiler/taco/. Accessed: (2020-02-19). 1, 3

[6] SAIPH DSL. URL https://pm.bsc.es/dsl/saiph/. Accessed: (2020-02-20). 3

[7] MLIR Google Group, . URL https://groups.google.com/a/tensorflow.org/d/forum/mlir. Accessed: (2020-02-21). 5

[8] Gitlab. URL https://about.gitlab.com/. Accessed: (2020-02-24). 5, 6

[9] LLVM-project Github, . URL https://github.com/llvm/llvm-project. Accessed: (2020-10-18). 38

[10] LLVM-project Github appropriate commit, . URL https://github.com/llvm/llvm-project/commit/11d2e63ab0060c656398afd8ea26760031a9fb96. Accessed: (2020-10-18). 38