An Intermediate Representation for Expressing and Optimizing Computations in Lattice Quantum Chromodynamics

by

Richard P. Sollee III

B.S., Computer Science and Engineering and Physics, MIT, 2023

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Richard P. Sollee III Department of Electrical Engineering and Computer Science May 10, 2024
Certified by:	Saman Amarasinghe Professor of Computer Science and Engineering, Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

An Intermediate Representation for Expressing and Optimizing Computations in Lattice Quantum Chromodynamics

by

Richard P. Sollee III

Submitted to the Department of Electrical Engineering and Computer Science
on May 10, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

The field of Lattice Quantum Chromodynamics faces massive scaling problems because of the large iteration spaces of the sums required which scale with the factorial of the number of atoms represented. The LQCD IR and rewrite system from this thesis allows tackling these scaling problems quicker and more effectively. The IR allows representing both mathematical concepts such as products and sums as well as algorithmic concepts such as precomputations. Our system requires minimal code to initialize the naive algorithm and apply effective rewrites to increase performance. This development time speedup allows trying various approaches with ease. The rewrite system allows correctness to be maintained at each step while being able to drastically change the algorithmic approach in search of better asymptotic bounds. Our approaches lead to up to 5x speedups and at worse 2x slowdowns for our most important problem, but with a better development cycle, requiring only 100s of SLOC compared to 1000s of SLOC.

Thesis supervisor: Saman Amarasinghe
Title: Professor of Computer Science and Engineering

Acknowledgments

I would like to thank professor Saman Amarasinghe for helping me find this amazing project to work on and for making the Compilers at MIT (COMMIT) group such an interesting community. I would also like to thank Mike Wagman and William Detmold for all their feedback and assistance with developing the system and how it could best be designed to help the physicists. I am also thankful for the entire COMMIT group for being so welcoming and providing interesting discussions which I got to appreciate for the last two and a half years.

I want to give a special thanks to Teo Collin who mentored me on this endeavor. Being able to work with him was a large motivator to staying on this project. His ability and willingness to answer my numerous questions while helping me learn more and develop my skills made this a great experience. His excitement for the work motivated me to strive for more and it encouraged me whenever we were met with roadblocks. This project began as my first UROP and stayed my only UROP which developed into this thesis because after working for a semester with Teo I knew this would be an amazing opportunity to learn from him.

I also want to thank my family, especially my parents, Dawn and Paul, and my sister Katie, for their support through my years at MIT. I would like to thank my sister's bunny Bufn for providing comfort when I faced problems. Lastly, I am grateful for my friends, especially those on the lightweight rowing team, for giving me their support and listening to me ramble about the various problems I faced on this project over the years.

Contents

Title page **1**

Abstract **3**

Acknowledgments **5**

List of Figures **11**

List of Tables **13**

1 Introduction **15**
1.1 Physics Background . . . 17
1.2 Computation Challenges . . . 18
1.3 Scheduling . . . 21

2 Related Works **23**
2.1 Classic DSLs . . . 23
2.2 Schedule Based DSLs . . . 24
2.3 Rewrite Based DSLs . . . 25

3 System Overview **26**
3.1 General Workflow . . . 26
3.2 Frontend Language . . . 26
3.3 LQCD IR . . . 30
3.4 Halide Scheduling File . . . 31

4 Our Intermediate Representation (LQCD IR) **32**
4.1 IR Overview . . . 32
4.2 IR Building Blocks . . . 32
4.2.1 Simple Example Programs . . . 35
4.3 Formalization . . . 37
4.3.1 Well Formed Sum . . . 37
4.3.2 Well Formed Index Choice . . . 37
4.3.3 Index Expression Ranges . . . 38
4.3.4 Permutation Index Uses . . . 38
4.3.5 Separate Indices . . . 38

4.3.6 Iteration Index Shorthand . . . 38
4.3.7 IR Isomorphism . . . 40
4.4 Semantics . . . 40
4.4.1 Sum Semantics . . . 40
4.4.2 Other Semantics . . . 41
4.5 IR manipulation tools . . . 41
4.5.1 Replacement Helpers . . . 43
4.5.2 Conversion Helpers . . . 43

5 IR Rewrites **45**
5.1 Motivation . . . 45
5.2 Separate Sum . . . 46
5.2.1 Motivation . . . 46
5.2.2 Algorithm . . . 47
5.3 Loop Linearization . . . 47
5.3.1 Motivation . . . 47
5.3.2 Algorithm . . . 49
5.4 Expression Partitioning . . . 50
5.4.1 Motivation . . . 50
5.4.2 Algorithm . . . 51
5.5 Expanding Permutations . . . 52
5.5.1 Motivation . . . 52
5.5.2 Algorithm . . . 52
5.6 Constant Propagation . . . 54
5.6.1 Motivation . . . 54
5.6.2 Algorithm . . . 55
5.7 Expression Merging . . . 55
5.7.1 Motivation . . . 55
5.7.2 Algorithm . . . 56
5.8 Precomputation over Ranges . . . 57
5.8.1 Motivation . . . 57
5.8.2 Algorithm . . . 58
5.9 Condense Choice . . . 59
5.9.1 Motivation . . . 59
5.9.2 Algorithm . . . 59

6 Case Studies **61**
6.1 Baryon . . . 61
6.1.1 Physics Setup . . . 61
6.1.2 Naive Code . . . 61
6.1.3 Rewrites Applied . . . 62
6.1.4 Analysis of Rewrite Impact . . . 63
6.2 Dibaryon-Dibaryon . . . 66
6.2.1 Physics Setup . . . 66
6.2.2 Naive Code . . . 67

6.2.3 Rewrites Applied . . . 67
6.2.4 Analysis of Rewrite Impact . . . 71
6.2.5 Impact . . . 72
6.3 Dibaryon-Hexaquark . . . 73
6.3.1 Physics Setup . . . 73
6.3.2 Naive Code . . . 75
6.3.3 Rewrites Applied . . . 75
6.3.4 Analysis of Rewrite Impact . . . 76
6.4 Hexaquark-Hexaquark . . . 76
6.4.1 Physics Setup . . . 76
6.4.2 Naive Code . . . 78
6.4.3 Rewrites Applied . . . 79

7 Future Work **80**
7.1 Automatic Algorithmic Optimization . . . 80
7.2 Automatic GPU Scheduling . . . 81

8 Conclusion **82**

A Large LQCD IR Printouts **84**
A.1 Baryon IR . . . 84
A.2 Dibaryon Dibaryon IR . . . 85
A.3 Dibaryon Hexaquark IR . . . 98
A.4 Hexaquark Hexaquark IR . . . 109

B Rewrite Code **118**
B.1 Dibaryon Dibaron Rewrites . . . 118

References **124**

List of Figures

1.1 SLOC for LQCD correlator computations 19

3.1 Diagram showing the work flow from LQCD problem definition to runnable code 27

4.1 Depicted is a slightly condensed version of the LQCD IR. 33
4.2 IR Formalization 39
4.3 Operational Semantics for IR to equivalent Python Semantics 42

5.1 Separate Sum Formalization The top covers not raising variables while the bottom covers raising variables 45
5.2 Separate Sum Examples 46
5.3 Loop Linearization Formalization 48
5.4 Expression Partitioning Formalization 50
5.5 Expanding Permutation Formalization 53
5.6 Constant Propagation Formalization 55
5.7 Expression Merging Formalization 55
5.8 Precomputation over Ranges Formalization 57
5.9 Condense Choice Formalization 59

6.1 Baryon Physics Setup 62
6.2 Comparision of various rewrites applied to the Baryon case. The figure shows the run times with respect to a multiplicative factor increase in the number of weights we use in the system. Loop linearization takes the scaling to $O(W)$ from $O(W^2)$ while the precomputations reduce scaling by a constant factor. . 64
6.3 Dibaryon Dibaryon Physics Setup 66
6.4 Timings for a scheduled Dibaryon-Dibaryon on different space sizes for a V100 72
6.5 Timings for a scheduled Dibaryon-Dibaryon on different space sizes for a A100 73
6.6 Dibaryon Hexaquark Physics Setup 74
6.7 Graphs showing how the GFLOPS and runtime change as the lattice size (N) increases for the Dibaryon Hexaquark case study. 77
6.8 Hexaquark Hexaquark Physics Setup 78
6.9 Comparision of SLOC for the physicist's Tiramisu code versus our system. The Dibaryon Dibaryon case also includes the SLOC for our GPU scheduling (the other cases were not GPU scheduled). 79

List of Tables

4.1 Overview of purpose of LQCD IR components 34

Chapter 1

Introduction

Lattice Quantum Chromodynamics (LQCD) allows us to predict and further understand fundamental aspects of our universe. It allows predicting behaviors that occur below atomic scales such as the strong force between gluons within the proton. Using computations following the theory of the Standard Model of particle physics, these results can be compared against experimental results to see how well the theory aligns with our observations of the universe. LQCD also allows predicting the behaviors of quarks in low energy states which are hard to experimentally obtain.

Lattice Quantum Chromodynamics faces massive scalability problems for computing desired Euclidean Correlation functions because the naive computation scales factorially in the number of quarks and polynomially in the spacial domain. LQCD problems allow modeling the dynamics of changes of quark properties such as position, flavor, and color. Modeling these involves simulating how every permutation of quarks interacts with every other permutation of quarks. Atoms even as small as carbon for example with 12 protons (and therefore 36 quarks) are too large to simulate with current methods given we would need to iterate over 36! permutations. Given a lattice of size N, the naive computation for a non-trivial example such as dibaryon-dibaryon has four nested loops over N as well as a nested loop over the 36 permutations and another two nested loops over values called the weights which

have a size on the order of 200. These nestings give at least seven nested loops which can lead to massive scaling in the naive case.

Unrolling the permutations of the system results in a typical Einstein summation. Using this, physicists have been able to hand optimize code to achieve better asymptotics and runtimes such as reducing the polynomial powers of some factors but these optimizations take thousands of lines of handwritten code in a Domain Specific Language (DSL) called Tiramisu [1] for every new problem they desire to solve. The code size blows up in a similar way to the size of the problem being simulated. This code blowup occurs because typical DSLs for loop scheduling do not feature ways of changing the algorithm of the program itself without modifying the code directly.

We seek a way of easing the process of running computations on new problems and optimizing the runtimes of these computations easily to produce competitive runtimes compared to prior methods while using significantly less code so that different problem sizes and configurations can be run with minimal setup effort. Our solution uses an intermediate representation (IR) to encapsulate the mathematics and precomputations of the program. Using this IR we can then perform rewrites on it to improve the runtime of the algorithm it represents.

The workflow from LQCD problem definition to executable code goes through three main stages: problem description, LQCD IR rewrites, and Halide scheduling [2]. A frontend system allows the physicists a intuitive way of describing the physics of the specific scenario which is then converted into our LQCD IR. The rewrites are then applied to optimize the algorithm. Finally a Halide function file is generated which is then updated with scheduling commands to target the hardware of the machine we run on, including GPUs. Our approaches leads to up to 5x speedsups and at worse 2x slowdowns for our most important problem, but with a better development cycle, requiring 100s of SLOC compared to 1000s of SLOC.

1.1 Physics Background

Quantum Chromodynamics is an area of theoretical physics which explores the results of the strong interaction between quarks. This field is able to help predict what happens within protons on the subatomic level [3]. Lattice Quantum Chromodynamics discretizes the space-time into a lattice which has a finite number of points and defines field values at each of these points[4]. Limiting the spacial locations ensures the math can remain tractable by avoiding infinities that appear when using continuous space at low energies [5]. In addition, limiting the number of spatial locations limits the degrees of freedom that need to be considered in the computations which allows computations to be done in physical situations where nonlinearity makes other methods hard or impossible.

Several key attributes make up the summations used in computing the Euclidean Correlation function of a system. The equations feature sums over space for every spacial location used (ie 2 locations with quarks means 2 loops over the size of the lattice) as well as 2 loops over a value called weights. The weights are used to allow accessing various indices of the arrays while tying those to a specific scaling factor. The sums also contain accesses into arrays based on the iteration values to retrieve the values to be multiplied. These accesses can be direct, where the iteration index is used to access part of the array directly, or indirect, where the iteration index is used as input to a function which maps to a much smaller range which is used to access part of the array.

Throughout this thesis we will use Equation 1.1 as a running example to show how different parts apply to our overall work. The structure of the equation matches that of the simpler LQCD problems closely because of the following components: loops over space (x, y), loops over weights (α, β), direct accesses to arrays, and indirect accesses to arrays. We also define f and g to map their domains to the numbers in the range 0 to $r - 1$.

$$\sum_{x,y}^{N} \phi(x)\phi(y) \sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \times S(x, f(\alpha), g(\beta)) \times S(y, f(\beta), g(\alpha)) \tag{1.1}$$

Prior work by physicists has allowed them to transform the runtimes of the weight loops from $O(W^2)$ to $O(W)$ (where W is the number of weights) as well as reduce situations with three loops over space to only two thereby taking the spacial loop asymptotics from cubic to quadratic. These improvements were based on the physical structure of the system. One example used precomputations over items in the formula relating to specific quarks to make precomputations that they named hadronic blocks[6]. Another paper which introduced a multi-baryon system used precomputations over the mathematical structure representing the baryons to make precomputations they call baryon blocks [7]. These optimizations had to be done by hand and required thousands of lines of code which makes these optimizations difficult to implement for larger problems. The scaling of these lines of code on different architectures can be seen in Figure 1.1.

1.2 Computation Challenges

The long runtimes for computing these results can be attributed to three primary characteristics: many loops over the lattice space, permutations of accesses needing to be considered, and indirect accesses, causing various problems in utilizing hardware architectures. Some of these challenges resemble those faced by the Tensor Contraction Engine (TCE)[8] such as our partitioning of expressions to reduce arithmetic and balancing the size of precomputations with executing equivalent math statements repeatedly. However, several of our challenges diverge from those handled by TCE such as out of order and indirect accesses in memory, iterations over permutation groups, and merging equivalent math expressions by finding the mapping of their isomorphism. These terms will be covered further on in chapter 4.

The naive version of some computations such as dibaryon-dibaryon require nested loops

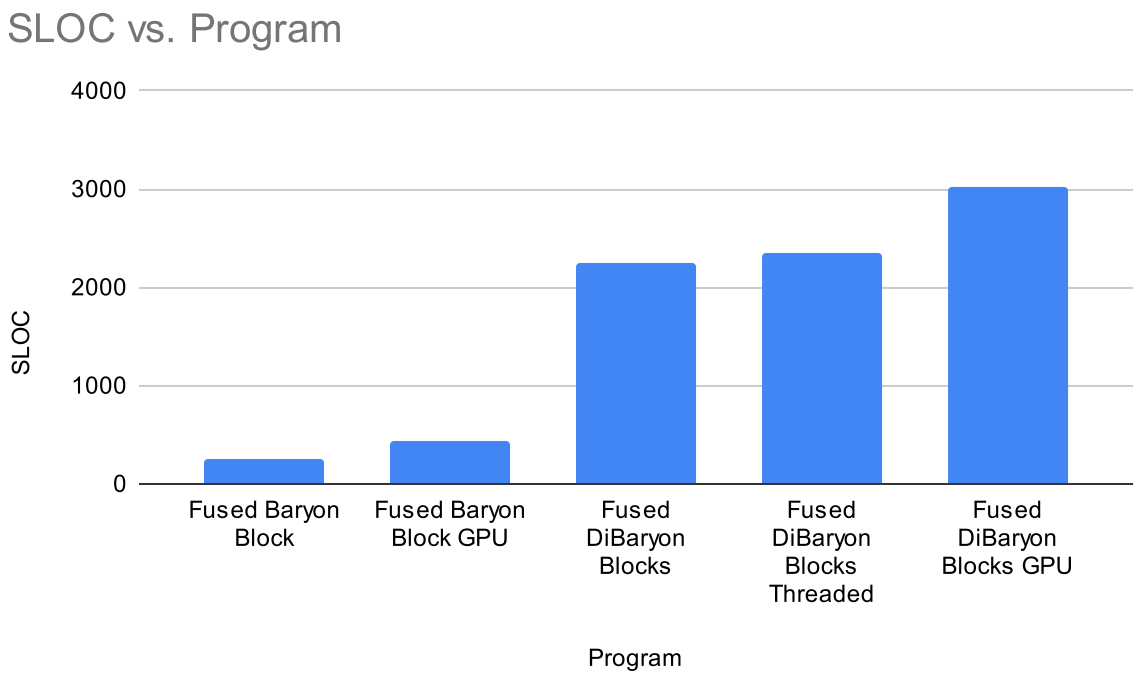


Figure 1.1: SLOC for LQCD correlator computations

that cause a quartic scaling with the size of the lattice which can cause runtimes to increase quickly given the large lattice sizes which are desired. Reducing these scalings requires finding the factors of the summand which are not dependent on the same indices among those being iterated over for the sum. Finding factors which depend on different indices allows applying rules governing summands and multiplications to have the corresponding factors iterated over separately instead of a nested fashion to reduce the scaling of the equation. The very simple example in Equation 1.2 can have its asymptotic bounds reduced though this process into Equation 1.3.

$$\sum_{x,y,z}^{N} F(x,z) \times F(y,z) \tag{1.2}$$

$$\sum_{z}^{N} \left(Y = \sum_{x}^{N} F(x,z) \textbf{ in } Y \times Y \right) \tag{1.3}$$

The equations describing the system require different permutations of the structure to be considered during computation which means that many different possible access functions to the data must be used at different points as shown in the example in Equation 1.4 where p determines what access function is applied on x to access F. These large numbers of combinations of accesses means that branching must occur to account for the possible accesses or that nonbranching code runs for each access pattern but then each access pattern is unable to make use of similar accesses in other sections. Each permutation being iterated over governs how the memory is being accessed and each individual instance of the permutation ordering can depend on different indices. These varying dependencies mean that one permutation order can depend on an index that another permutation order does not use. In order to take advantage of the potential speedups which are possible for each permutation ordering, the permutations may need to be expanded to the indices they depend on (meaning we unroll the corresponding loop) so the structure of the equation can be determined. Determining the structure allows finding what permutation orderings have isomorphic structures and then applying the proper rewrites to speed up each equation as much as possible given its unique structure.

$$\sum_{x}^{N} \sum_{p}^{3} F([\alpha(x), \beta(x), \gamma(x)][p]) \tag{1.4}$$

Many of the accesses to the data in memory are determined by taking the iteration variable and mapping it to a different value which leads to non-consecutive memory accesses thereby causing slowdowns. Arranging the loop orderings and some of the precomputations can help alleviate this potentially but overall the indirect accesses still pose a major slowdown.

1.3 Scheduling

A program can be separated into two distinct but related parts: the algorithm and the schedule. The algorithm is responsible for what loops need to be done, what floating point operations need to be done, and other similar operations. The schedule is responsible for defining how the results of the operations of the algorithm are stored and in what order the operations occur (including optimizations such as loop tiling)[2].

By separating the algorithm from the schedule, it becomes easier to optimize because you can reorder the scheduling while leaving alone the code defining the algorithm. Many domain specific languages (DSLs) such as Halide and Tiramisu take this approach to representing a program.

Domain Specific Languages (DSLs) are a powerful tool which allow succinctly expressing problems within the domain in which the language is designed to be used. The more limited feature set of a DSL compared to a full programming language allows for the DSL to leverage the context of the domain to create optimizations which can not be assured to be correct in a traditional programming language such as the scheduling commands described above.

The original physicist code took advantage of Tiramisu [1], a polyhedral compiler, to write their algorithms and do scheduling for the program. While Tiramisu allowed flexibility of loop structures, it lacked the same level of support as other compilers and DSLs. This lack of support and better understanding of the use cases for the LQCD computations has lead to Halide, a programming language for image processing pipelines[2], being seen as a better fit here with better community support. As mentioned above, Halide allows separating the algorithm from the scheduling commands which allows approaching our optimizations at different levels.

While these DSLs can allow some ease of optimizations by separating the schedule and algorithm, they prevent expressing some optimizations. The issue stems from the inability to change the algorithm when attempting to schedule the program. Some optimizations require

the algorithm itself to be changed to allow a desired scheduling, which these DSLs do not support doing without directly modifying the code creating the initial algorithm.

Chapter 2

Related Works

The past research most related to this project lies in the field of compilers and Domain Specific Languages (DSLs). DSLs allow describing problems efficiently in their respective domains and providing optimizations based on these domains and other principles to generate efficient code easily, like we hope to do with LQCD. These DSLs can be roughly categorized into three types: classic, schedule based, and rewrite based.

2.1 Classic DSLs

Classic DSLs provide little to no control to the user and act as a classic compiler. The steps in using these DSLs involve simply describing the problem in the DSL and then letting the DSL compile the result. A simple example could be an APL (Array Processing Language) compiler where you describe your algorithm in APL and it compiles it by applying all the optimizations it deems possible [9]. One similar to our work, called the Tensor Contraction Engine (TCE), involved taking quantum chemistry problems, representing them in an intermediate representation, and applying automatic optimizations to them. Their work focused on getting the problems easily specified in a general tensor representation which allowed general compiler loop optimizations for tensors to occur with some domain specific optimizations [8]. SQL is another simple example commonly used in databases where queries on

the database are expressed in the language and the query engine decides the most optimal way of executing them [10].

2.2 Schedule Based DSLs

Schedule based DSLs have an additional step in their process where you add scheduling to the algorithm after describing the algorithm in the DSL and before compiling. Scheduling commands typically encompass processes such as memory management, loop optimizations, and the use of accelerators, such as GPUs. As mentioned in section 1.3, Halide allows for defining one's algorithm, primarily through loop structures and operations, and then scheduling the order of the loops and operations separately to add optimizations such as tiling [2]. A simple example from their tutorials can be seen in Listing 2.1 where a function is defined and the loop ordering is changed. Halide also has support for autoscheduling which allows one to create their algorithm in Halide and then have the autorscheduler pick a good schedule without needed to have a deep understanding of the architecture [11]. The Tiramisu compiler, also mentioned above, supports more complex iteration spaces than Halide but lacks the same level of support and maintenance of the codebase [1]. TACO is a tensor algebra compiler which allows for defining tensor algebra expressions and having the computations scheduled based on the density of the tensors involved. Later work called WACO automatically schedules the computation and storage format in TACO based on the sparsity pattern[12]. Taichi was written to assist in 3D visual computing to take advantage of sparsity that occurs in those domains but allow the user to define their own data structures[13]. With images and graphs, Opt allows for least squares problems in graphics which can be used to easily generate different implementations of the desired function with varying tradeoffs so that options can be explored easily [14]. GraphIt works in a similar way to Halide but for graphs where the user can define the algorithm over a graph that they want and then separately add scheduling like parallelism [15].

```
import halide as hl
x, y = hl.Var("x"), hl.Var("y")
gradient = hl.Func("gradient")
gradient[x, y] = x + y
gradient.reorder(y, x) # reorder the loops
output = gradient.realize([4, 4])
```

Listing 2.1: Halide example (simplified tutorial 5 from the website)

2.3 Rewrite Based DSLs

Rewrite based DSLs have an additional step in their process where you can apply rewrites to the algorithm after describing the algorithm in the DSL and before compiling. These rewrites allow manipulating the algorithm while ensuring correctness. An early example of this pattern is the Elevate strategy language that can be used to modify an algorithm written in Rise, a functional language similar to Halide [16]. The Elevate language allows modifying the algorithm itself in ways a schedule cannot which they demonstrate in their paper by contrasting their work with TVM [17], a schedule based DSL that is very similar to Halide. Being able to prove the soundness of rewrites is desirable leading to the development of ATL, a framework written in Coq, which allows justifying the correctness of transformations while being able to achieve schedules similar to those of Halide [18]. SPIRAL allows writing floating-point code targeting parallel platforms by using a rewrite system to ensure correctness between the kernel that the user creates and the resulting program that gets run [19]. Spiral is an older stlye compared to the above because the rewriting is done automatically most of the time, meaning the rewriting is not considered part of the programming process.

Chapter 3

System Overview

3.1 General Workflow

The workflow from LQCD problem definition to executable code goes through three main stages seen in Figure 3.1. The problem definition is first described using a frontend language created to allow the physicists a intuitive way of describing the physics of the specific scenario. This frontend is then converted into the **LQCD IR** which forms the crux of our process and is described in chapter 4. Once converted to the **LQCD IR**, the algorithm can be transformed using rewrites described in chapter 5. Once the rewrites are applied, we proceed to the third stage of a generated Halide function file. Once this file is generated, Halide scheduling commands are added to the file to allow optimally targeting the hardware of the machine we run on. Finally, the Halide file runs to carry out the computation and deliver a result.

3.2 Frontend Language

Our frontend language models LQCD problems at a level understood by physcists. Though this is not the focus of this thesis, we will describe it briefly here. We will attempt to give intuition for what the naive programs will look like based on a few objects.

At the top level of an LQCD program, a physicist declares three types of sizes and

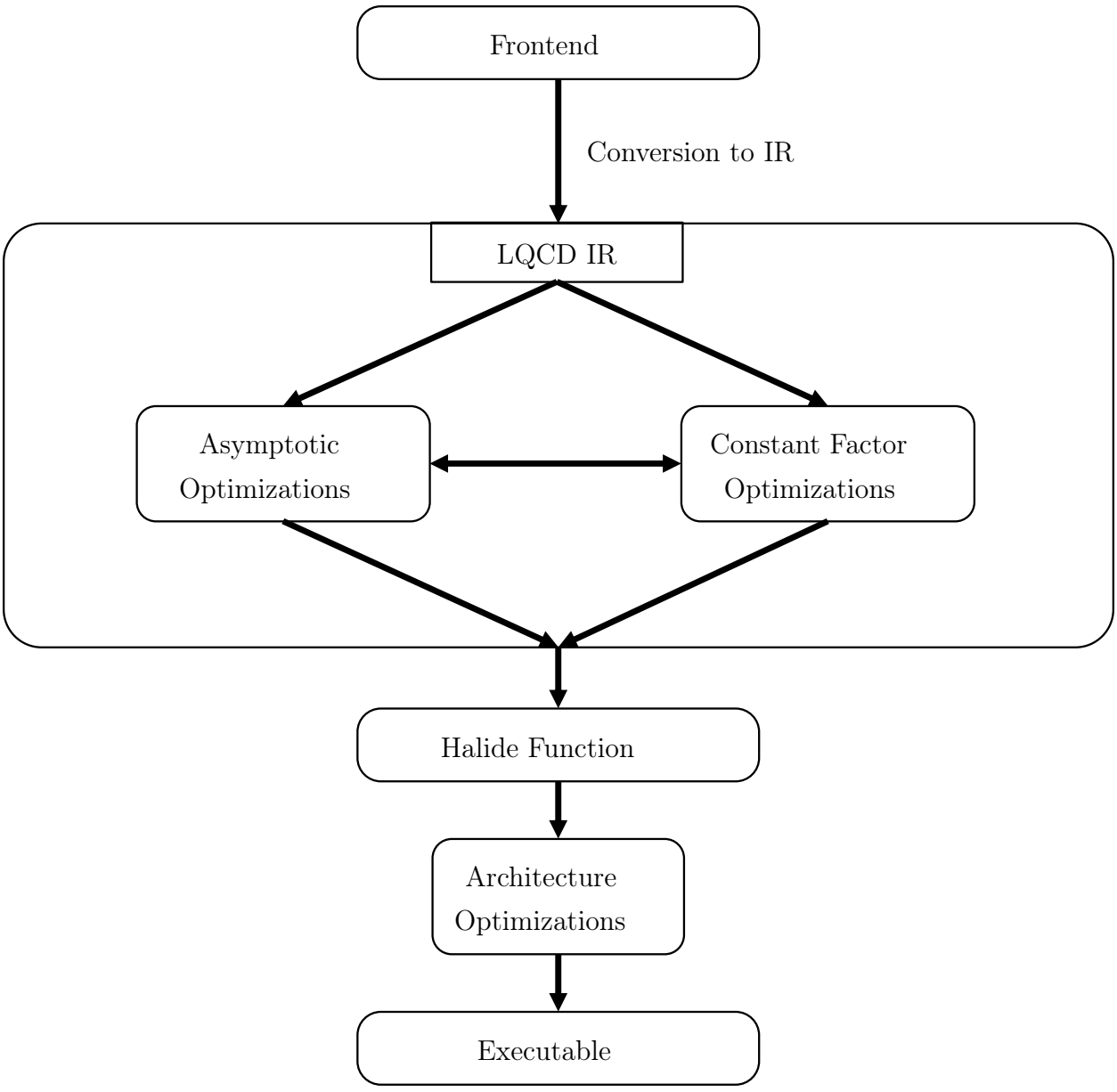


Figure 3.1: Diagram showing the work flow from LQCD problem definition to runnable code

indices for these sizes: external, lattice, and rank. External indices will iterate over different problems while lattice and rank indices will form a top level reduction. So the naive program will be a loop over the range of the external indices followed by a reduction over all rank and space indices. At the bottom of this loop nest, lies one more group of loop nest and a single statement, that are determined by the final inputs to the program: the structure of the quarks and the structure of the weights.

The physicist inputs the structure of the quarks as a list of n regular and n anti quarks of a given flavor (effectively an enum type in our system), a map from a quark to a lattice index, and a flavor by flavor matrix that describes what quarks can interact with each other. Our system forms group of permutations that ensures that every quark of a given flavor can interact with every eligible anti-quark of a given flavor. This determines the last loops: they are a reduction over all of these permutations. This also determines that there will be n accesses to the propagator, an (at minimum) 8 dimensional complex tensor that is flavor by flavor by lattice by lattice by spin by color by spin by color. The specific meanings of spin and color do not matter here, but the n access to the propagator will be determined by a quark and a permutation of this quark into an anti-quark. Each access will use the space and flavor correspond to the quark and anti-quark pair of the permutation. The $2n$ spin and color accesses in this product of n propagator accesses at the bottom of the permutation, rank, lattice, and external loops will be determined by the final input: the weight tensors.

Lastly, the physicist inputs a list of weight tensors: these are tensors that are associated to quarks, lattice indices, external indices, and rank indices. A single access to each weight tensor will be part of the product of propagators determined above. The meaning of the latter three objects is then clear: the weight tensor will be accessed by these indices. The quark associations is less clear. Each quark can only be associated to one weight and this association implies an additional input to the generated program: two indirection maps, one for spin and one for color, that are accessed by some subset of the indices associated to the tensor and that are used to access the spin and color components of the propagator. Thus,

the accesses to the spin/color of the propagator are indirection maps accesses determined by the quark and the permuted anti-quark as well as the weight tensors descriptions. These indirection thus tie together the weight tensors with the product of propagators.

To summarize our final program, we note that it is a description of the initial rank, lattice, and external indices/sizes, as well as the structure of the quarks and weights in this program. To summarize the output program, we note that the program is a loop over external indices, followed by a reduction over lattice, external, and permutation indices. Finally, we note that a single statement lies in this reduction: a product of accesses to the propagator and of accesses to the weight tensors where the accesses are determined by the quark structure and the indices associated to the weights. The frontend is evolving as we understand physicists better so this description may change and we do not commit to a full description here. We provide this description to give a sketch of the types of programs we will see represented in the LQCD IR. An example is shown in Listing 3.1 and the resulting naive IR is Listing 6.2.

```
N = latticeSize("N")
src = latticeIndex("s", True, N)
snk = LatticeIndex("s", False, N)
u1 = quarkField(u, src)
d1 = quarkField(d, src)
u2 = quarkField(u, src)
ubar1 = quarkField(ubar, snk)
dbar1 = quarkField(dbar, snk)
ubar2 = quarkField(ubar, snk)
wRnkSize = size("w_src_1_rank")
wRnkSrc = rankIndex("w_src_1_rank", wRnkSize)
wRnkSnk = rankIndex("w_src_2_rank", wRnkSize)
spatialWeightsSrc = spatialWeights("psi", [src])
quarkWeightsSrc = quarkWeightVector("w_src", [u1, d1, u2], rankIdxs=[wRnkSrc], indexRankIdxs
    =[wRnkSrc])
spatialWeightsSnk = spatialWeights("phi", [snk])
quarkWeightsSnk = quarkWeightVector("w_snk", [ubar1, dbar1, ubar2], rankIdxs=[wRnkSnk],
    indexRankIdxs=[wRnkSnk])
srcField = Field("Baryon", [u1, d1, u2], [spatialWeightsSrc, quarkWeightsSrc])
snkField = Field("antiBaryon", [ubar1, dbar1, ubar2], [spatialWeightsSnk, quarkWeightsSnk])
qS = Prop("S")
propIndex = [L(False), C(3, False), S(4, False),L(True), C(3, True), S(4, True)]
prop = DiagonalProp([qS, qS], [u, d], propIndex)
computation = inner(srcField, prop, snkField)
```

Listing 3.1: Baryon Frontend

3.3 LQCD IR

While the **LQCD IR** is described in detail in chapter 4, we will have a short overview here of its use in the system as a whole. The **LQCD IR** is primarily composed of `<expr>`s, `<index>`es, and `<indexExpr>`s. The various `<expr>`s such as `Sum`, `VarAccess`, and `Mult` represent mathematical or algorithmic expressions we can carry out to compute our result. The `<index>`es represent the domains we must iterate over in our `Sum`s to compute the results. The `<indexExpr>`s allow describing how we can take an `<index>` we iterate over and use it or modify it to perform a lookup on a variable in the program.

The LQCD problems we face can be translated into summations over various indices with summands that are a product of accesses to variables. In addition, some of these variables have accesses using an intermediate mapping from the iteration index to a different range. The described parts above of `<expr>`s, `<index>`es, and `<indexExpr>`s allow us to express these aspects of the computation and additional parts such as the `Let <expr>` allow denoting optimizations such as precomputations.

The primary goal of the **LQCD IR** in the system is to allow applying rewrites to modify the algorithm we want to run while ensuring we maintain correctness. Therefore, the main power of the **LQCD IR** comes from the rewrites we apply to it seen in chapter 5. The rewrites can be categorized into those which offer speedups to the algorithm and those which make the IR simpler to read or add additional substeps in the computation which can be useful when scheduling on hardware. Of the rewrites which offer speedups, they can be grouped into asypmtotic optimizations, which change the O runtime of the program, and constant factor optimizations, which change the scaling factor of the O runtime.

3.4 Halide Scheduling File

Generating a Halide scheduling file allows us to have a rerunable file to execute the problem but does not require running the entire pipeline every time one wants to run the computations. Most importantly, the Halide scheduling file allows one to add Halide scheduling instructions on top of the algorithm generated in Halide by the LQCD IR so that one can get maximal output from their machine. The user wants to consider optimizations such as GPU scheduling, cache use for precompute sizes, and parallelism to get faster speeds.

Chapter 4

Our Intermediate Representation (LQCD IR)

4.1 IR Overview

To represent the mathematical structure and optimizations of our program, we define an intermediate representation we call the **LQCD IR**. In designing the IR, we needed to consider the mathematical operations that were necessary as well as algorithmic steps such as precomputations. This lead to adding mathematical constructs such as `Sum`, `Mult`, and `Conj` as well as algorithmic and memory steps such as `Let` and `VarAccess`. These computations also involve many forms of indirect memory accesses so the `<indexExpr>` type was needed to encapsulate all the possibilities of accesses while the `<index>` type encapsulates all the different types of loop iterations, namely loops over a linear range and loops over permutation groups.

4.2 IR Building Blocks

The main building block of the program is the `<expr>` which can represent many different operations. A `Sum` defines a looping stucture to generate a tensor with free indices (which

⟨*perm*⟩ ::= Sym(int k)
  | Cross(⟨*perm*⟩* perms)
  | Stab(⟨*perm*⟩ perm, int k, ⟨*indexExpr*⟩ v)

⟨*indexSize*⟩ ::= ConstantSize(int size) | PermSize(⟨*perm*⟩ perm)

⟨*index*⟩ = (name iname, ⟨*indexSize*⟩ high_val)

⟨*indexExpr*⟩ ::= Index(⟨*index*⟩ access)
  | ConstIndex(int access)
  | IndexFunc(⟨*var*⟩ var, ⟨*indexExpr*⟩* access)
  | IndexChoice(⟨*indexExpr*⟩* vars, ⟨*indexExpr*⟩ access)
  | PermIndex(⟨*index*⟩ perm, ⟨*indexExpr*⟩ access)

⟨*var*⟩ = (name vname, int num_dim)

⟨*assign*⟩ = (⟨*var*⟩ var, ⟨*expr*⟩ rhs, ⟨*index*⟩* lhs)

⟨*expr*⟩ ::= Sum(⟨*index*⟩* free_indices, ⟨*index*⟩* iter_indices, ⟨*expr*⟩ summand)
  | Mult(⟨*expr*⟩* exprs)
  | Add(⟨*expr*⟩* exprs)
  | Const(float val)
  | Conj(⟨*expr*⟩ to_conj)
  | Let(⟨*expr*⟩ let_expr, ⟨*var*⟩ lname, ⟨*expr*⟩ use_expr, ⟨*index*⟩ * lhs)
  | MultiLet(⟨*assign*⟩* assigns, ⟨*expr*⟩ use_expr)
  | VarAccess(⟨*var*⟩ v, ⟨*indexExpr*⟩* indices)
  | ExprChoice(⟨*expr*⟩* exprs, ⟨*indexExpr*⟩ access)
  | Sign(⟨*index*⟩ perm)
  | Det(⟨*var*⟩ v, int size, ⟨*indexExpr*⟩* above)

Figure 4.1: Depicted is a slightly condensed version of the LQCD IR.

LQCD IR Type	**Purpose**
Sum	Define a summation with free indices that define the resulting dimensions and iteration indices which we perform a reduction over
Mult/Add	Perform multiplication/addition operations between expressions
Const	Represent a constant value
Conj	Obtain the complex conjugate of an expression
Let/MultiLet	Allow performing one or more precomputations which are then used in another expression
VarAccess	Lookup values from memory based on `<indexExpr>`s
Sign	Get the sign of a current iteration of a permutation
ExprChoice	Allow selecting which expression to use depending on an `<indexExpr>` value
`<index>`	Define iteration spaces over fixed ranges or permutation groups
Index	Used to have an `<indexExpr>` where the value depends on an `<index>` being iterated over
ConstIndex	Used to define an `<indexExpr>` with a fixed value
IndexFunc	Allow a memory lookup based on an `<indexExpr>` to be used as its own `<indexExpr>`
IndexChoice	Allow selecting which `<indexExpr>` to use depending on an `<indexExpr>` value
PermIndex	Allow accessing a part of the permutation of the current iteration (ex: accessing the 0th index of the current iteration of the symmetric group 3)
Sym	Define a symmetric permutation group iteration
Stab	Stabilize a permutation group at a given index to be a value dependent on an `<indexExpr>`
Cross	Define a permutation iteration space that is the cross product of other permutations

Table 4.1: Overview of purpose of LQCD IR components

could be used to access the result) and reduction indices (which are looped over and also fed to the summand <expr>). Mult and Add are those respective n-ary operations between their children. Const represents constant numbers. Conj indicates the result needs to be conjugated (we are operating on complex numbers). Sign allows accessing the sign of a permuation which is being iterated over. Let and MultiLet represent precomputing an expression (or expressions), assigning it to a variable (or variables), and evaluating the use expression with the new variables defined. VarAccess allows indexing into an input or Let variable with given <indexExpr>s. Det is computing the determinant of a section on a matrix variable. ExprChoice allows choosing what expression to evaluate depending on an <indexExpr>.

For specifying iterations and accesses to variables we define <index> and <indexExpr> respectively. The <index> is used for specifying variables that loop with value 0 up to <indexSize> − 1. The <indexExpr> is used for accessing values dependent on the iteration state. Index is used to access with a given <index> value defined in a Sum. ConstIndex is a constant access into a variable. IndexFunc accesses a variable with an <indexExpr> and uses that result as its value. IndexChoice, similar to ExprChoice, chooses what <indexExpr> to use depending on a given <indexExpr> value. In addition to normal iterations over a range, we allow iteration over a permutation group. A Sym <perm> represents an iteration over a symmetric group of a given size. A Stab is when we fix a specified index of the permutation to a value given by an <indexExpr>. A permutation iteration is created when the <indexSize> is a PermSize and is used as an access with PermIndex by taking a PermSize type <index> and an <indexExpr> to choose which value of the current permutation to use as the value.

4.2.1 Simple Example Programs

Below are several example programs written in the **LQCD IR** to show basic behaviors of the system. The matrices_match function runs the generated IR given as the first argument with the inputs of the second argument and ensures the result matches the third argument.

Listing 4.1 shows that the number of iterations that occur over a permutation is the correct size of the corresponding symmetric group. Listing 4.2 ensures that the sign of each permutation instance is correct as this property shows up in our computations. Listing 4.3 shows how a simple matrix multiplication can be executed in the IR.

```
perm_size = 5
perm_sum = LQCD_IR.Sum(
    [], [LQCD_IR.index('x', LQCD_IR.PermSize(LQCD_IR.Sym(5)))], LQCD_IR.
   Const(1.0)
)
expected = factorial(perm_size)
matrices_match(perm_sum, {}, expected)
```

Listing 4.1: Permutation Size Example (the number of iterations should match the factorial of the size of the symetric group)

```
perm_size = 5
perm_index = LQCD_IR.index('x', LQCD_IR.PermSize(LQCD_IR.Sym(perm_size)))
perm_sum = LQCD_IR.Sum(
    [], [perm_index], LQCD_IR.Mult(LQCD_IR.Const(1.0), LQCD_IR.Sign(
   perm_index))
)
matrices_match(perm_sum, {}, 0)
```

Listing 4.2: Permutation Sign Example (the signs should cause the result to sum to 1)

```
# inp_ABCD is a dictionary mapping 'A' and 'B' to numpy arrays of
   dimentions (MATRIX_SIZE, MATRIX_SIZE)
MATRIX_SIZE = 10
MATRIX_SIZE_IR = LQCD_IR.ConstantSize(MATRIX_SIZE)
A_var, B_var = LQCD_IR.var("A", 2), LQCD_IR.var("B", 2)
i_ind, j_ind, k_ind = LQCD_IR.index("i", MATRIX_SIZE_IR), LQCD_IR.index("j
   ", MATRIX_SIZE_IR), LQCD_IR.index("k", MATRIX_SIZE_IR)
i_ind_expr = LQCD_IR.Index(i_ind)
j_ind_expr = LQCD_IR.Index(j_ind)
```

```
k_ind_expr = LQCD_IR.Index(k_ind)
mat_mul_AB = LQCD_IR.Sum(
    [i_ind, j_ind],
    [k_ind],
    LQCD_IR.Mult(
        LQCD_IR.VarAccess(A_var, [i_ind_expr, k_ind_expr]),
        LQCD_IR.VarAccess(B_var, [k_ind_expr, j_ind_expr]),
    ),
)
expected = np.linalg.multi_dot([inp_AB['A'], inp_AB['B']])
matrices_match(mat_mul_AB, inp_AB, expected)
```

Listing 4.3: Matrix Multiplication Example

4.3 Formalization

In this section, we will be defining several terms for IR structures to make references to them later when discussing the rewrites that have been implemented. In addition we will define how we describe parts of the IR in latex mathematically.

4.3.1 Well Formed Sum

When we depict a `Sum` in LaTeX we will have the free indices on top and the iteration indices on the bottom. We define a well formed `Sum` (WFS) as a `Sum` where its iteration indices and free indices are disjoint and all free indices appear in the parent sum if there is a parent (as shown in Figure 4.2a).

4.3.2 Well Formed Index Choice

We define a well formed `IndexChoice` to be an `IndexChoice` where the range of the access `<indexExpr>` (which we define in Figure 4.2c) is the same as the number of `IndexChoice`

options. This definition can be seen in Figure 4.2b.

4.3.3 Index Expression Ranges

For the various `<indexExpr>` types defined in the **LQCD IR**, we define a range property (rng) which gives the size of the range so that an `<indexExpr>` with range k can only output integers $[0, k)$. These range calculations for the options can be seen in Figure 4.2c.

4.3.4 Permutation Index Uses

In some cases, the **LQCD IR** type requires an `<index>` but it actually requires a more strict type of an `<index>` with a high_val of type `PermSize`. This occurs with `PermIndex` and `Sign` as seen in Figure 4.2d. These cases require a permutation index because their behavior is defined with respect to a given permutation. For example, a `PermIndex` accesses an index of the array representing the current permutation and `Sign` represents the sign of a permutation so neither of these operations would be valid on a non-permutation.

4.3.5 Separate Indices

Many times we have a set of indices which we need to separate into two disjoint sets so we define this behavior in Figure 4.2e.

4.3.6 Iteration Index Shorthand

We often use a permutation of a symmetry group of some size k so to shorten the notation we define a notation symPerm shown in Figure 4.2f. We also often use the iteration over a constant size k so to shorten the notation we define a notation constIter shown in Figure 4.2g.

$$\frac{X = \sum_I^J \quad I \cap J = \emptyset}{X \vdash \text{WFS}}$$

$$\frac{X = \sum_I^J \quad X \vdash \text{WFS} \quad Y = \sum_K^L X \quad K \cap L = \emptyset \quad J \subseteq K \cup L \quad I \nsubseteq K \cup L}{Y \vdash \text{WFS}}$$

(a) Well Formed Sum (WFS) Formalization

$$\frac{\text{type } j = \texttt{<indexExpr>}, \text{rng } j = n \quad \forall J_i, \text{type } J_i = \texttt{<indexExpr>}}{I \vdash \text{WFIC } I \ J_i} \text{ WFIC}$$

(b) Well Formed Sum `IndexChoice` Formalization

$$\frac{i = \texttt{<index>}(\text{high\_val} = \texttt{ConstantSize}(r_i)) \quad I = \texttt{Index}(i)}{I \vdash \text{rng } I = r_i}$$

$$\frac{I = \texttt{ConstIndex}(k)}{I \vdash \text{rng } I = k}$$

$$\frac{\exists j \text{ s.t. type } j = \texttt{<indexExpr>} \quad X = \texttt{var s.t. } \max X = k \quad I = \texttt{IndexFunc}(X, j)}{I \vdash \text{rng } I = k}$$

$$\frac{\text{WFIC } I \ J_i \quad \forall J_i, \text{rng } J_i = r_i}{I \vdash \text{rng } I = \max r_i}$$

(c) `<indexExpr>` ranges

$$\frac{\texttt{Sign}(i)}{i = \texttt{<index>}(\text{high\_val} = \texttt{PermSize})} \qquad \frac{\texttt{PermIndex}(i, j)}{i = \texttt{<index>}(\text{high\_val} = \texttt{PermSize}),\ j = \texttt{<indexExpr>}}$$

(d) PermIndex uses

$$\frac{I = \text{set of } \texttt{<index>} \text{ s.t. } \|I\| \geq 2}{\exists K, L \text{ s.t. } K \cup L = I, K \cap L = \emptyset, \|K\| \geq 1, \|L\| \geq 2} \text{ indSep I K L}$$

(e) Separating Indices into Disjoint Sets

$$\frac{X = \text{symPerm } k}{X = \texttt{<index>}(\text{high\_val} = \texttt{PermSize}(\texttt{Sym}(k)))} \text{ symPerm k}$$

(f) symPerm Notation

$$\frac{X = \text{constIter } k}{X = \texttt{<index>}(\text{high\_val} = \texttt{ConstantSize}(k))} \text{ constIter k}$$

(g) constIter Notation

Figure 4.2: IR Formalization

4.3.7 IR Isomorphism

We define two IR structures to be isomorphic (notated as iso AB for A and B being isomorphic) using a conversion from the IR to a graph structure. If the graph structures are isomorphic then we have an isomorphism of the IR structure. Converting the IR elements to a graph involves generating a tree structure with `Const`, `Sign`, and `VarAccess` being the leaves. To convert an IR `<expr>` to the graph structure, we first create a node representing the current expr. Then if it is not a `Const`, `Sign`, or `VarAccess`, we recursively create the nodes for each of its children in the IR and add edges between the current node and the children. Next we label the current node with its type (ex: `Sum`, `Const`, etc.). If the current node is a `VarAccess`, we also label the node with the ranges of each of the accesses. This labeling ensures an isomorphic mapping needs to map nodes to the same types and the `VarAccess`es have the same dimensions.

4.4 Semantics

We will now define how to convert a given LQCD IR `<expr>` to an example equivalent program. While the actual conversion to Halide is slightly different, the algorithmic structure of what generates the final answer is the same. The conversion process is recursive and uses the IR's tree-like structure. Some operational semantics of conversion to python code are shown in Figure 4.3. The translations make use of Γ to represent context mappings of names to values.

4.4.1 Sum Semantics

Given a `Sum`, for every free `<index>` and iteration `<index>`, that we have not already created a for-loop for at a higher level, we create a new for loop for each `<index>`. Before the loops we define a variable with all the free indices as dimensions and initialize all accesses to 0.

In the body of the innermost loop, we evaluate the summand with a recursive call having the `<index>`es now defined with their current loop value. We then access the result variable at the points defined by the current free indices and add to it the result of the evaluated summand.

4.4.2 Other Semantics

`Add`, `Mult`, `Const`, `Conj`, and `Sign` all trivially do the actions expected based on their definition after having recursively evaluated their children. When encountering a `Let`/`MultiLet`, shown in Figure 4.3c, you create the let variable and assign it the result of recursing on the let expression. Then the use expression runs with the let variable now defined in its environment. This precomputation can occur within the nested loopings of `Sum`s and `Let`s. For a `VarAccess`, shown in Figure 4.3a, the `<indexExpr>` for each access is evalutated and then the results are used to access the memory location of the `<var>` specified. For an `ExprChoice` or an `IndexChoice`, the `<indexExpr>` for the access is evalutated and then the `<expr>`/`<indexExpr>` at the selected index of the options is evaluated.

4.5 IR manipulation tools

To ease the process of creating the rewrites and choosing specific locations of the IR to apply them, many helper functions were created. For the IR we define a `key` to a location in the IR to be a list of integers where the first integer of the list tells which constructor child to traverse down. This leads to a recursive key structure for identifying nodes where an empty list identifies the current node, the list $[0]$ selects its zeroth child (which depends on the type of the current node), the list $[0, 2]$ first selects the zeroth child then the second child of that node, and so on.

$$\frac{n \in Z_{>0} \quad \Gamma \vdash \texttt{var}(\text{'x', n}) \to v \quad \Gamma \vdash \forall k \in \{1, \ldots, n\}, i_k \to j_k}{\Gamma \vdash \texttt{VarAccess}(\texttt{var}(\text{'x', n}), [i_1, \ldots, x_n]) \to v[j_1, \ldots, j_n]}\ \text{py}$$

(a) VarAccess translation

$$\begin{aligned}
&\text{LOOP-SUM}(f, i_0, \ldots, i_k, j_0, \ldots, j_n) = \\
&\qquad \text{result}[:, \ldots, :] = 0 \\
&\qquad \text{for } y0 \text{ in range(rng } j_0) : \\
&\qquad\quad \ldots \\
&\qquad\quad\quad \text{for } yn \text{ in range(rng } j_k) : \\
&\qquad\quad\quad\quad \text{for } x0 \text{ in range(rng } i_0) : \\
&\qquad\quad\quad\quad\quad \ldots \\
&\qquad\quad\quad\quad\quad\quad \text{for } xn \text{ in range(rng } i_n) : \\
&\qquad\quad\quad\quad\quad\quad\quad \text{result}[y0, \ldots, yn] \mathrel{+}= f(x0, \ldots, xk, y0, \ldots, yn)
\end{aligned}$$

$$\frac{X = \sum_I^J \text{summand} \quad I = \{i_1, \ldots, i_k\} \quad J = \{j_1, \ldots, j_n\} \quad \Gamma \vdash \text{summand} \to f(\ldots)}{\Gamma \vdash X \to \text{LOOP-SUM}(f, i_0, \ldots, i_k, j_0, \ldots, j_n)}\ \text{py}$$

(b) Sum translation

$$\frac{I = \texttt{IndexChoice}([J_0, \ldots, J_{n-1}], j) \quad X = \textbf{Let } \Upsilon(J) = F(J) \textbf{ in } P \quad \Gamma \vdash P \to p(\ldots)}{\Gamma \vdash X \to \Upsilon = \text{py } F(J); p(\ldots)}\ \text{py}$$

(c) Let translation

Figure 4.3: Operational Semantics for IR to equivalent Python Semantics

4.5.1 Replacement Helpers

There are three main helpers we defined which allow carrying out an action involving a location in an IR tree defined by a given `key`. The helper `get_from_ir` takes in a tree and `key` and returns the IR node located at the given position in the tree. The helper `replace_in_ir` takes in a tree, `key`, and new value and replaces the `key` position in the tree with the new value provided. These functions are very helpful for extracting parts of the IR, rewriting them, and placing them back in the structure. This process was repeated often enough to define another helper, `run_on_loc`, which takes in a tree, `key`, and a function. This helper extracts the node at the given `key` position in the tree, then passes it to the provided function, and finally takes the result of that function and places the result in the tree at the `key` location. This helper makes rewrites much simpler to apply by allowing simply specifying the location to apply the rewrite and passing the rewrite function or a lambda function that uses it.

We have also defined a `find_and_replace_in_ir` helper which takes in a tree, a needle function, and a replacement function/value. This helper allows defining a function to apply to all parts of the tree and if the function returns true then the location in the tree is replaced. If the replacement type is simply a value, then the locations where the needle returns true have that new value placed. If the replacement type is a function, then the node which passed the needle function is given to the replacement function and the return value is placed at the original location. This flexibility of a replacement function allows having the replacement value depend on the current nodes complete properties which makes it very flexible for use in rewrite rules.

4.5.2 Conversion Helpers

For ease of dealing with the trees created by `Mult` and `Add`, helpers have been created to convert the nodes to arrays and back. The `get_mult_tree` and `get_add_tree` take in

`Mult` and `Add` nodes respectively and return an array of all the children which are being multiplied/added. This eases the process of iterating through items being multiplied/added and factorizing them during rewrites. To convert back, the functions `prod_ir` and `sum_ir` take in an array of nodes and return an IR node that has the items of the array all being multiplied/added.

Chapter 5

IR Rewrites

5.1 Motivation

To allow changing the algorithmic structure of a given computation, but ensure correctness at intermediate steps, we define functions called rewrites which take in a given LQCD IR representation and return a new modified version which may be algorithmically different but is mathematically equivalent. By having intermediate steps ensuring correctness it allows for piecing together different rewrites to get the desired end structure while easing the process of debugging the intermediate steps.

$$\frac{X = \sum_I^J \quad \text{indSep } I\,K\,L}{X \to \sum_K^J \sum_L^{J\cup K}} \text{ sepSum K}$$

$$\frac{X = \sum_I^J F(I) \quad \text{indSep } I\,K\,L \quad F(I) = G(K)H(L)}{X \to \sum_K^J G(K) \sum_L^{J\cup K} H(L)} \text{ sepSumMove K}$$

Figure 5.1: Separate Sum Formalization
The top covers not raising variables while the bottom covers raising variables

$$\sum_{x,y} F(x)G(y) \rightarrow \tag{5.1}$$

$$\sum_{x} F(x) \sum_{y} G(y) \tag{5.2}$$

$$\sum_{x}^{N} \phi(x) \sum_{y}^{N} \phi(y) \sum_{\alpha}^{W} \sum_{\beta}^{W} w(\alpha)w(\beta) \times S(x, f(\alpha), g(\beta)) \times S(y, f(\beta), g(\alpha)) \tag{5.3}$$

Figure 5.2: Separate Sum Examples

5.2 Separate Sum

5.2.1 Motivation

The separating sum rewrite allows taking a `Sum` which has many iteration indices (representing reduction domains) and splitting it into a sum of a sum which together encapsulate the original iteration indices (formalized in Figure 5.1). This operation allows other rewrites which must analyze all the iteration indices of a given sum to be applied where it could not before because the sum had iteration indices that would interfere with the desired rewrite. The rewrite also has the option of moving accesses to variables which depend only on the outermost sum to be above the innermost sum which can help reduce the number of FLOPs which occur. Equation 5.1 and Equation 5.2 show a simple before and after of applying this rewrite to a sum with moving the variables where we can see that the sum is now two sums and variables dependant only on the raised sum have been moved. Equation 5.3 shows the result of this rewrite being applied to the first and second sums of Equation 1.1 (which represent two nested loops each). The x and y sums have now been separated and the move vars command was true so the $\phi(x)$ access has been moved upwards. For the α and β sum, the sum was simply separated with no variables being moved. There is the same amount of total nesting but with moving variables the location of some computations has been moved and the structure of the IR was changed even in the case of not moving variables.

5.2.2 Algorithm

To apply the rewrite, we take the original sum and the iteration indices we want to raise. Given these, we create a new sum with the same free indices but with the iteration indices as the ones we specified to raise. We then take the original sum and move the specified iteration indices to be free indices. We then set the modified original sum as the summand of the new sum and return the new sum. If we also desired to raise any variable accesses fixed by the raised iteration indices then we would search the original summand for factors which are fixed by the raised indices and remove them. We would then take these expressions and have the new sum's summand now be a `Mult` of these additional expressions and the modified original sum, instead of just the modified original sum.

5.3 Loop Linearization

5.3.1 Motivation

The Loop Linearization rewrite allows us to take a `Sum` with multiple nested loops and unnest them so that the iterations over each loop are additive instead of multiplicative. This linearization allows for substantial asymptotic speed increases given how much the quadratic scaling potentially grows over the liner scaling. Given the rewrite formalization depicted in Figure 5.3 , we can describe the speedups saying that we have a `Sum` with N iteration indices with a maximum range of n which are used in at most M `IndexFunc`s each with the maximum range of any `IndexFunc` being bounded by r. These parameters give the initial runtime as $O(n^N)$ which the loop linearizing can reduce to $O(r^{NM}Nn)$ and since in practice $r^{NM} << n$ and $N << n$ we are able to take a non-linear polynomial scaling down to linear. Equation 5.4 and Equation 5.5 (where r is the range of the β function) show the before and after of applying the loop linearization to a sum where you can see the sums become serial instead of nested. Equation 5.6 shows loop linearization applied to the inner sum of

$$\frac{L_i \subseteq L \quad \dfrac{Y = F(X_i(L_i), \dots)}{\forall i, X_i = \texttt{<indexExpr>}, L_i = \text{ set } \texttt{<index>}} \quad \exists j, k \; L_j \cap L_k = \emptyset}{\vdash \text{indepIndExprs } FXL} \text{ indepIndExprs}$$

G is an expression depending on indices,

the I_i are the disjoint indices,

$X$is the groups of index expressions with defining

X_i in this function being the index expressions which depend on only I_i

and a_i are the set of constIter's over the ranges of the index expressions in $X_j, j > i$

LOOP-LIN$(G, I_0, \dots, I_n, X) =$

$$\textbf{Let } V_0(a_0) = \sum_{I_0}^{a_0} G(a_0, X_0(I_0))$$

$$\textbf{in Let } V_1(a_1) = \sum_{I_1}^{a_1} V_0(a_1, X_1(I_1))$$

$$\vdots$$

$$\textbf{in } \sum_{I_n} V_n(X_n(I_n))$$

$$\frac{L_i \subseteq I \cup J \quad Y = \sum_I^J G, \text{indepIndExprs } GXL \quad \exists i \text{ s.t. } \text{ rng } X_i(L_i) < \text{ rng } L_i}{\dfrac{\text{multIndSep } I \; I_i, \forall i \exists j \text{ s.t. } L_j \subseteq I_i}{Y \to \text{LOOP-LIN}(G, I_0, \dots, I_n, X)}}$$

Figure 5.3: Loop Linearization Formalization

Equation 1.1. As you can see in the example where linearizing reduction was done to the weights iterations, we reduce the runtime from $O(N^2W^2)$ to $O(N^2W)$ (the r factor is left out because it is a constant).

$$\sum_{x,y} F(\alpha(x))G(\beta(y)) \rightarrow \tag{5.4}$$

$$\textbf{Let } D(b) = \sum_{x} F(\alpha(x))G(b) \quad b \in \{0, \ldots, r-1\} \textbf{ in } \sum_{y} D(\beta(y)) \tag{5.5}$$

$$\sum_{x,y}^{N} \phi(x)\phi(y) \Bigg(\textbf{Let } D(k,m) = \sum_{\beta}^{W} w(\beta) \times S(x,k,g(\beta)) \times S(y,f(\beta),m) \quad k,m \in \{0,\ldots,r-1\} \textbf{ in } \sum_{\alpha}^{W} w(\alpha) \times D(f(\alpha),g(\alpha)) \Bigg) \tag{5.6}$$

5.3.2 Algorithm

To apply loop linearizing, we first get all the index expressions used to access indices of variables. Then we group them into disjoint sets based on what iteration indices they depend on. We then separate the expressions in the summand into groups that depend on just one set (call these E_i with i being the ordering we assign to the indices) and a group that holds all expressions that depend on more than one set E_s. We then choose an ordering of which sets of indices to iterate over, prioritizing iterating over the smaller sets first. Given the first set to iterate over, we take the expressions E_0 and E_s and replace all the index expressions from the other sets with iterations over their ranges. These iterations over ranges define the free indices of our precomputation and the iteration indices are the first set we chose to iterate over. For the use statement of this precomputation we then recursively define other precomputations where we access the already made precomputation with the original index

$$\frac{I = \text{set of } \texttt{<index>} \quad \|I\| \geq 2 \quad I = \cup_i I_i \quad \forall I_i I_k \, i \neq j I_i \cap I_k = \emptyset}{\vdash \text{multIndSep } I \; I_i} \text{ multIndSep}$$

$$\frac{X = \sum_I^J C(I \cup J) \quad \text{multIndSep } I \; I_i \quad C(I \cup J) = \prod_i A_i(I_i \cup (J_i \subseteq J))}{X \to C_i = \sum_{I_i}^{J} A_i(I_i \cup (J_i \subseteq J)), \prod_i C_i}$$

Figure 5.4: Expression Partitioning Formalization

expressions that were replaced at that level and keep the range replacements for levels that are not iterated over yet. This results in a final use statement that simply iterates over the final unused indices and accessed the last precomputation with the index expressions that depended on that final set of indices.

5.4 Expression Partitioning

5.4.1 Motivation

Expression partitioning allows reducing asymptotics of a sum by partitioning the summands into two or more summands which are able to be summed up separately and then have the results multiplied together to get the original result. Being able to sum up the parts in sequential steps instead of nested loops gives large asymptotic improvements which in the dibaryon-dibayon case can be used to turn some of the quartic expressions to quadratic and the rest of the quartic expressions can become cubic. Figure 5.4 shows the criteria for the rewrite and the general result. Equation 5.7 and Equation 5.8 show the before and after of applying the expression partitioning to a sum where you can see the sums become serial instead of nested which reduces the scaling. Given the example in Equation 5.9, we can apply expression partitioning to the inner sum to achieve Equation 5.10 which has two consecutive loops over space instead of the two nested ones present in the original equation.

$$\sum_{x,y} F(x)G(y) \rightarrow \tag{5.7}$$

$$\textbf{Let } C = \sum_{x} F(x), D = \sum_{y} G(y) \textbf{ in } C \times D \tag{5.8}$$

$$\sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \sum_{x,y}^{N} \phi(x)\phi(y) \times S(x, f(\alpha), g(\beta)) \times S(y, f(\beta), g(\alpha)) \tag{5.9}$$

$$\begin{aligned} \textbf{Let } D(\alpha,\beta) &= \sum_{x}^{N} \phi(x) \times S(x, f(\alpha), g(\beta)) \quad \alpha,\beta \in \{0,\ldots,W-1\} \\ E(\beta,\alpha) &= \sum_{y}^{N} \phi(y) \times S(y, f(\beta), g(\alpha)) \quad \alpha,\beta \in \{0,\ldots,W-1\} \\ &\textbf{in } \sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \times D(\alpha,\beta) \times E(\beta,\alpha) \end{aligned} \tag{5.10}$$

5.4.2 Algorithm

Once we have identified a `Sum` that is ready for partitioning we take the following steps. For each expression in the product of the summand we determine what iteration indices it depends on. Using these dependencies we then find the disjoint sets of iteration indices which we can partition into. For each disjoint set we create a precomputation using the iteration indices of the disjoint set and the expressions in the original summand that depend on those indices. After creating all the precomputations, we create the use expression which multiplies the results of the precomputations together.

5.5 Expanding Permutations

5.5.1 Motivation

Applying this rewrite (as shown in Figure 5.5) allows the different structures of the individual permutation iterations to be viewed and optimized based on their individual structures. Viewing each permutation iteration can allow more optimizations because different iterations may have different structures that require unique optimizations that could not be applied to the original summand as a whole.

Applying the rewrite means that we can unroll the loop over the permutations to allow constant accesses where there were previously permutation accesses and replaces sign of permutation expressions with a constant value. This allows other rewrites to then be applied to find groupings between different permutations. A simple before and after given by Equation 5.11 and Equation 5.12 shows how the permutation loop becomes unrolled and the permutation accesses are now `IndexChoice` accesses. Given the example in Equation 5.13, we can apply permutation expansion to the innermost sum to achieve Equation 5.14 as another example.

$$\sum_{p}^{\text{symPerm } 2} \sum_{x,y} F([x,y][p[0]])G([x,y][p[1]]) \rightarrow \tag{5.11}$$

$$\textbf{Let } C = \sum_{x,y} F([x,y]\big[[0,1][0]\big])G([x,y]\big[[0,1][1]\big]), D = \sum_{x,y} F([x,y]\big[[1,0][0]\big])G([x,y]\big[[1,0][1]\big]) \textbf{ in } C \times D \tag{5.12}$$

5.5.2 Algorithm

To apply the rewrite, we take the given sum and permutation index and loop over the permutation size. For each permutation, we create a new expression where the uses of the

$$\frac{p = \text{symPerm } k \qquad X = \texttt{Sign}(p) \qquad i \in \{0, 1, \ldots, k! - 1\}}{\text{fixPerm } X\, i = \text{sign}(S_k, \text{iteration } i)}\ \text{fixPermSign}$$

$$\frac{j = \texttt{<indexExpr>} \qquad p = \text{symPerm } k \qquad X = \texttt{PermIndex}(p, j) \qquad i \in \{0, 1, \ldots, k! - 1\}}{\text{fixPerm } X\, i = \texttt{IndexChoice}(S_k \text{ iteration } i \text{ values as list}, j)}\ \text{fixPermPermIndex}$$

$$\frac{X = \sum_I^J Y \qquad \exists i \in I \text{ s.t. } i = \text{symPerm } k}{X \to \textbf{Let } Y_0(I \cup J - i) = \text{fixPerm } Y\, 0, \ldots, Y_{k!-1}(I \cup J - i) = \ldots \textbf{ in } Y_0 + \cdots + Y_{k!-1}}$$

Figure 5.5: Expanding Permutation Formalization

permutation index have been replaced with an index choice into a list of constant accesses and the signs are replaced with the computed permutation sign of the current permutation. We then create a `MultiLet` where the let expressions are all the expressions generated for each permutation and the use expression is an addition of all these precomputes (as seen in Figure 5.5).

$$\sum_{x,y}^{N} \phi(x)\phi(y) \sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \sum_{p}^{\text{symPerm } 2} S([f(\alpha), g(\alpha)][p[0]], [g(\beta), f(\beta)][p[1]]) \times S([f(\beta), g(\beta)][p[0]], [g(\alpha), f(\alpha)][p[1]]) \tag{5.13}$$

$$\begin{aligned}
\textbf{Let } D(\alpha, \beta) =& S([f(\alpha), g(\alpha)][[0,1][0]], [g(\beta), f(\beta)][[0,1][1]]) \times \\
& S([f(\beta), g(\beta)][[0,1][0]], [g(\alpha), f(\alpha)][[0,1][1]]) \quad \alpha, \beta \in \{0, \ldots, W-1\} \\
E(\beta, \alpha) =& S([f(\alpha), g(\alpha)][[1,0][0]], [g(\beta), f(\beta)][[1,0][1]]) \times \\
& S([f(\beta), g(\beta)][[1,0][0]], [g(\alpha), f(\alpha)][[1,0][1]]) \quad \alpha, \beta \in \{0, \ldots, W-1\} \\
\textbf{in } & \sum_{x,y}^{N} \phi(x)\phi(y) \sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \times (D(\alpha, \beta) + E(\beta, \alpha))
\end{aligned} \tag{5.14}$$

5.6 Constant Propagation

5.6.1 Motivation

This rewrite allows simplifying expressions to be able to compare them to other expressions and find similarities in their structure. As shown in Figure 5.6, we can apply the propagation when a choice has a constant access (so we can do the access already) or all the possible access results are the same (so the actual choice made does not change the result). Equation 5.12 provides a great example to show constant propogation in action as this rewrite ties well with loop unrolling. We can see the result of this in Equation 5.15. Equation 5.14 gives another great example to apply constant propagation to where Equation 5.16 shows the result of applying the constant propagation there.

$$\textbf{Let } C = \sum_{x,y} F(x)G(y), D = \sum_{x,y} F(y)G(x) \textbf{ in } C \times D \tag{5.15}$$

$$\begin{aligned}
\textbf{Let } D(\alpha,\beta) =& S(f(\alpha), f(\beta)) \times \\
& S(f(\beta), f(\alpha)) \quad \alpha,\beta \in \{0,\ldots,W-1\} \\
E(\beta,\alpha) =& S(g(\alpha), g(\beta)) \times \\
& S(g(\beta), g(\alpha)) \quad \alpha,\beta \in \{0,\ldots,W-1\} \\
\textbf{in } & \sum_{x,y}^{N} \phi(x)\phi(y) \sum_{\alpha,\beta}^{W} w(\alpha)w(\beta) \times (D(\alpha,\beta) + E(\beta,\alpha))
\end{aligned} \tag{5.16}$$

$$\frac{i = \texttt{IndexChoice}(J, g) \qquad g = \texttt{ConstantIndex}(k)}{i \to J[k]}\ \text{constPropIndexChoice}$$

$$\frac{X = \texttt{ExprChoice}(J, g) \qquad g = \texttt{ConstantIndex}(k)}{X \to J[k]}\ \text{constPropExprChoice}$$

$$\frac{i = \texttt{IndexChoice}(J, g) \qquad J = \{j\}}{i \to j}\ \text{IndexChoiceAllSame}$$

$$\frac{X = \texttt{ExprChoice}(J, g) \qquad J = \{j\}}{X \to j}\ \text{ExprChoiceAllSame}$$

Figure 5.6: Constant Propagation Formalization

$$\frac{X = \textbf{Let } Y_0(I_0) = Z_0, \ldots, Y_{n-1}(I_{n-1}) = Z_{n-1} \textbf{ in } G(Y_0, \ldots, Y_{n-1}) \qquad G = \texttt{<expr>} \qquad \forall i, j \text{ iso } Z_i\ Z_j}{X \to \textbf{Let } F(k, I_0) = \text{merged } Z_0, \ldots, Z_n \textbf{ in } G(F(0), \ldots, F(n-1))}$$

Figure 5.7: Expression Merging Formalization

5.6.2 Algorithm

To apply the rewrite we check if there are any of the given situations described above. If there is an `IndexChoice` or an `ExprChoice` with a constant access we replace the choice with the value at the chosen index of the options. If there is an `IndexChoice` or an `ExprChoice` with all matching options we replace the choice with the first option (since they are all matching). This process is repeated until no part matches the criteria for rewriting.

5.7 Expression Merging

5.7.1 Motivation

This rewrite allows simplifying the generated representation which results in simpler Halide code. A simpler representation eases the process of checking the IR for issues. Additionally, simpler Halide code generation allows quicker compilation and an easier process for scheduling the Halide because of fewer duplicate statements that still must be scheduled separately. A first simple example of before and after given in Equation 5.17 and Equation 5.18 shows

how two precomputations can become merged. We can also see the use of this process by applying expression merging to Equation 5.10. The result can be seen in Equation 5.19 where there is no longer a need for multiple let statements as the D and E let expressions could be combined.

$$\textbf{Let } C = \sum_x F(x), D = \sum_y F(y) \textbf{ in } C \times D \rightarrow \tag{5.17}$$

$$\textbf{Let } C = \sum_x F(x) \textbf{ in } C \times C \tag{5.18}$$

$$\begin{aligned} \textbf{Let } D(\alpha,\beta) = \sum_x^N \phi(x) \times S(x, f(\alpha), g(\beta)) \quad \alpha,\beta \in \{0,\ldots,W-1\} \\ \textbf{in } \sum_{\alpha,\beta}^W w(\alpha)w(\beta) \times D(\alpha,\beta) \times D(\beta,\alpha) \end{aligned} \tag{5.19}$$

5.7.2 Algorithm

We are only able to merge precomputations which are isomorphic as defined in subsection 4.3.7 and shown in this rewrite in Figure 5.7. To apply the rewrite, we take all the precomputations of the `MultiLet` and recursively apply the rewrite to an array of all the precomputation expressions. In general, the process of merging the expressions consists of comparing the children of the expression and recursively merging each of those and remaking the parent with the merged children. When attempting to merge two incompatible parts, such as indices with different ranges, an error is thrown. Otherwise, if the pieces are compatible but not exact matches or matches given an isomorphism mapping then we create an `IndexChoice` or an `ExprChoice` depending on the type being merged.

Finding the isomorphisms between expressions happens when merging `Mult` expressions. To merge a `Mult`, we first break each `Mult` into an array of its factors. We then fix the

$$\frac{L_i \subseteq I \cup J \qquad X = \sum_I^J G(X_0(L_0), \ldots, X_{k-1}(L_{k-1})) \qquad \exists i \text{ s.t. } \text{rng } X_i(L_i) < \text{ rng } L_i}{\frac{X \to \textbf{Let } Y(a_0, \ldots, a_{k-1}) = G(a_0, \ldots, a_{k-1}) \textbf{ in } \sum_I^J Y(X_0(L_0), \ldots, X_{k-1}(L_{k-1}))}{a_i = \text{constIter } (\text{rng } X_i(L_i))}}$$

Figure 5.8: Precomputation over Ranges Formalization

first `Mult` given and do no reorderings on it. All comparisons happen in relation to the first `Mult`. For each other expression, we iterate through the permutations of orderings of the factors and evaluate how well the given ordering matches and merges with the ordering of the first `Mult`. The best match is chosen for each `Mult` we want to merge and then used for the recursive merge. The mappings of the indices from the previous ordering to the new ordering are then noted for merging any parent `Lets`. To merge `Let` expressions, any reorderings of the indices from reordered `Mult` children are noted and the accesses of the `Let` variable are rewritten so that all of the ranges of the corresponding indices match between the `Let` expressions that are being merged.

5.8 Precomputation over Ranges

5.8.1 Motivation

Precomputation over ranges allows reducing the number of FLOPs in the resulting code. This occurs when the initial code has multiple iterations that map to the same summand result because of `<indexExpr>`s which map to the same result given different inputs. By computing these summands ahead of time, we can simply look up the summand instead of taking FLOPs to compute the result every time. Equation 5.20 and Equation 5.21 show a simple before and after of performing this precomputation. Additionally, Equation 5.22 shows the precomputation applied to the inner sum of Equation 1.1. As you can see in the example, parts of the original summand are now precomputed ahead of time and accessed in the main sum.

$$\sum_{x,y} F(\alpha(x))G(\beta(y)) \rightarrow \tag{5.20}$$

$$\textbf{Let } D(a,b) = \sum_{x} F(a)G(b) \quad a,b \in \{0,\ldots,r-1\} \textbf{ in } \sum_{x,y} D(\alpha(x),\beta(y)) \tag{5.21}$$

$$\sum_{x}^{N}\sum_{y}^{N} \phi(x)\phi(y)\Bigg(\text{Let } D(k,m,l,p) = S(x,k,m)\times S(y,l,p) \quad k,m,l,p \in \{0,\ldots,r-1\} \\ \sum_{\alpha}^{W}\sum_{\beta}^{W} w(\alpha)w(\beta) \times D(f(\alpha),g(\beta),f(\beta),g(\alpha))\Bigg) \tag{5.22}$$

5.8.2 Algorithm

To apply the optimization we form a precomputation over the ranges of all the `IndexFunc`s and then access that precomputation in the main summand (shown in Figure 5.8). To do this, we first find all the `<indexExpr>`s used. Then we create a new `<index>` for each one which loops over the range of the `<indexExpr>`. The summand is then modified so that the use of each unique `<indexExpr>` is replaced with an `Index <indexExpr>` which uses the `<index>` created for the range of the original `<indexExpr>`. This summand is then moved to a let expression with a binding corresponding to the new `<index>`es used in the modified summand. The summand of the `Sum` is then replaced with a `VarAccess` where the accesses are the original `<indexExpr>`s being used in the access that represents the range `<index>` it was replaced with in the precomputation. This `Sum` is then placed in the use expression of the `Let` and the new `Let` is returned.

$$\frac{X = \textbf{Let } Y(a_i, b, \dots) = G(F, \dots) \textbf{ in } Y(a_i, b, \dots) \qquad F = \texttt{IndexChoice}([a_1, a_2, a_1, a_2, \dots], b)}{X \to \textbf{Let } Y(c, b, \dots) = G(\texttt{IndexChoice}([a_1, a_2], c), \dots) \textbf{ in } Y(\texttt{IndexChoice}([0, 1, 0, 1 \dots], b), \dots)}$$

Figure 5.9: Condense Choice Formalization

5.9 Condense Choice

5.9.1 Motivation

Condense choice allows reducing the number of precomputations that need to be done. As seen in Figure 5.9, we can perform a condense choice when an `IndexChoice` has several values that are repeated as options. A precomputation would initially form a separate storage location for each of those choices despite some being equivalent. Moving the selection of all the options to the use statment allows the let statement to just precompute over the differing options and then all the reused options can be accessed from the use statement. Provided a very simple example in Equation 5.23, we can see what applying the condense choice does to the a index in Equation 5.24.

$$\textbf{Let } D(a) = F([0, 1, 0, 1][a]) \quad a \in \{0, 1, 2, 3\} \quad \textbf{in } \sum_{c,d}^{4} D(c) \times D(d) \tag{5.23}$$

$$\textbf{Let } D(\alpha) = F([0, 1][\alpha]) \quad \alpha \in \{0, 1\} \quad \textbf{in } \sum_{c,d}^{4} D([0, 1, 0, 1][c]) \times D([0, 1, 0, 1][d]) \tag{5.24}$$

5.9.2 Algorithm

To apply the optimization we give a fixed size index which is used in the `IndexChoice`es we would like to condense and find the `Let` which uses it. We then find all the `IndexChoice`es in the let which use the given index. Then a list is formed of all the unique options in the

`IndexChoice`. The original index choice is then replaced with an `IndexChoice` over these options. Finally, the use statement is updated so that references to the `Let` which used the original fixed size index now access the `Let` using an `IndexChoice` which maps the original index to the smaller range of the corresponding unique option we desire.

Chapter 6

Case Studies

These case studies together represent the code needed to implement the correlation functions done in previous 2 nucleon studies [7]. The most important is the dibaryon dibaryon which is 95% of the optimized runtime, so that one will have the most real data.

6.1 Baryon

6.1.1 Physics Setup

The physical situation of the baryon system features two lattice sites: one source and one sink. The source node has three quarks: two up and one down. The sink node has three quarks: two up and one down and all the quarks having antiness. These are represented in Figure 6.1.

6.1.2 Naive Code

The naive code generated features two nested loops over space (one for each lattice site), three nested weight loops, and two nested permutation loops for a total of seven main nested reduction domain loops. One of the permutation loops is over the symmetric group 1 and the other is over the symmetric group 2. The summand consists of a product of three accesses to

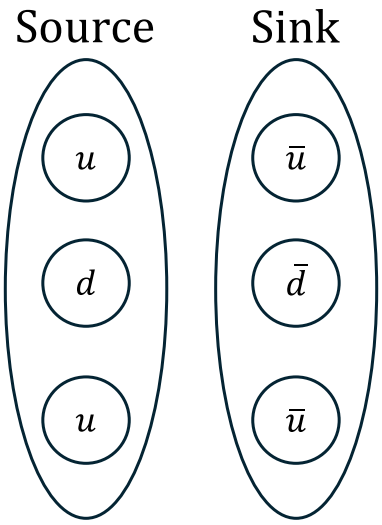


Figure 6.1: Baryon Physics Setup

the propogator (which has six dimensions), two accesses to spacial weights, and two accesses to other weights.

6.1.3 Rewrites Applied

The IR starting point can be seen in Listing 6.2. To begin we first want to simplify out the iteration over the symetric group 1. We can do this using a couple steps. First, we retrieve the index which represents it (named "down" in our code). Then we expand out the permutation which creates a `MultiLet` with one assign. Finally we can take the let statement and replace all the times the variable is referenced in the use statement with the actual let value (can be though of inlining the variable where it is used). Finally we apply `remove_unnecessary_index_choice` which cleans up our `IndexChoice`es. This process makes the IR cleaner without the excess index.

Looking at the IR now, there are no ways to reduce the loops over space by expression partitioning but we can reduce the weight looping with loop linearization. To prep it for loop linearization we need to isolate the weight iterations and the remainig permutation iteration from the rest of the loops. Therefore, we apply `simplify_conj`, which moves a conj of an `Add`/`Mult` into its children instead. This allows separating spacial accesses from

the weight accesses later since they are no longer combined together under a `Conj`. Then we apply `separate_sum` where we raise up the space indices. Then on the inner sum we apply `loop_linearize`. The rewritten IR can be seen in Listing A.1.

```
def baryon_rewrite(comp):
    down_perm = find_index_by_name(comp, 'down')
    comp = push_use_into_let(expand_perm(comp, down_perm))
    comp = remove_unnecessary_index_choice(comp)
    comp = separate_sum(simplify_conj(comp, ["w_src", "w_snk"]), ["s_src",
    "s_snk"], move_vars=True)
    comp = run_on_loc(comp, [2, 2, 0, 0], loop_linearize)
    return comp
```

Listing 6.1: Baryon Rewrites

6.1.4 Analysis of Rewrite Impact

The main result of the rewrites is the loop strengthening which takes the asymptotic scaling from $O(W^2)$ to $O(W)$ where W is the number of weights given to the system. These results can be seen in Figure 6.2 where we have four different rewrite possibilities applied and show how the run times increase as the number of weights given to the system is scaled by a multiplicative factor. The naive line shows no rewrites applied while the loop linearized line shows the scaling difference which occurs after the loop linearizing rewrite. While we do not go over the application of the precomputation rewrite in the previous section (though the rewrite itself is covered in section 5.8), we also have data from applying that rewrite to both the naive case and the loop linearized case. In both situations it reduces the scaling by a constant multiplicative factor so we see a slight speed up.

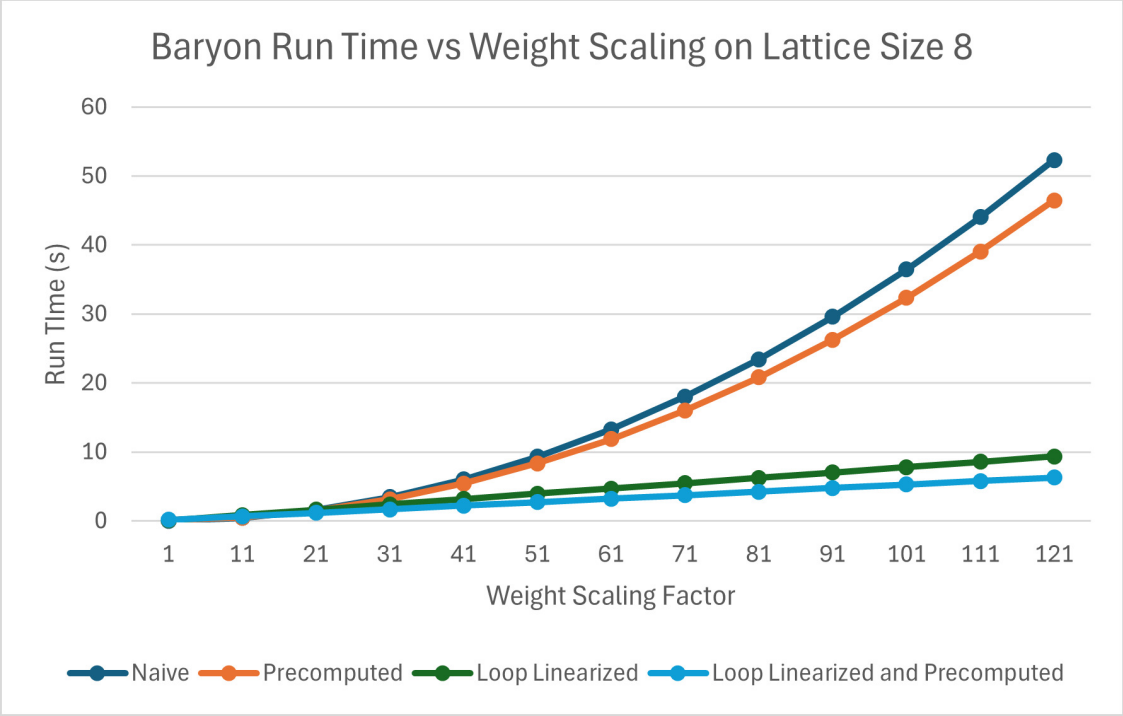


Figure 6.2: Comparision of various rewrites applied to the Baryon case. The figure shows the run times with respect to a multiplicative factor increase in the number of weights we use in the system. Loop linearization takes the scaling to $O(W)$ from $O(W^2)$ while the precomputations reduce scaling by a constant factor.

```
output_152: NDArray = np.zeros(1, )
summand_1 = 0.0
for (w_src_2_rank_136, s_src_116, s_snk_118, w_src_1_rank_134) in zip(range(w_src_1_rank),
    range(N), range(N), range(w_src_1_rank)):
    for up_148 in itertools.permutations(range(2)):
        for down_150 in itertools.permutations(range(1)):
            summand_1 += (
                S_138[s_src_116, w_src_spin_0_120[w_src_1_rank_134],
                      w_src_color_0_120[w_src_1_rank_134],
                      [s_snk_118, s_snk_118][up_148[0]], [
                          w_snk_spin_0_126[w_src_2_rank_136],
                          w_snk_spin_2_130[w_src_2_rank_136]
                      ][up_148[0]], [
                          w_snk_color_0_126[w_src_2_rank_136],
                          w_snk_color_2_130[w_src_2_rank_136]
                      ][up_148[0]]] *
                S_138[s_src_116, w_src_spin_2_124[w_src_1_rank_134],
                      w_src_color_2_124[w_src_1_rank_134],
                      [s_snk_118, s_snk_118][up_148[1]], [
                          w_snk_spin_0_126[w_src_2_rank_136],
                          w_snk_spin_2_130[w_src_2_rank_136]
                      ][up_148[1]], [
                          w_snk_color_0_126[w_src_2_rank_136],
                          w_snk_color_2_130[w_src_2_rank_136]
                      ][up_148[1]]] * sign(up_148) *
                S_138[s_src_116,
                      w_src_spin_1_122[w_src_1_rank_134],
                      w_src_color_1_122[w_src_1_rank_134],
                      [s_snk_118][down_150[0]],
                      [w_snk_spin_1_128[w_src_2_rank_136]][
                          down_150[0]],
                      [w_snk_color_1_128[w_src_2_rank_136]][
                          down_150[0]]] * sign(down_150) *
                (psi_140[s_src_116] * w_src_142[w_src_1_rank_134])
                * np.conj((phi_144[s_snk_118] *
                           w_snk_146[w_src_2_rank_136])))
output_152[None] = summand_1
```

Listing 6.2: Baryon Naive IR

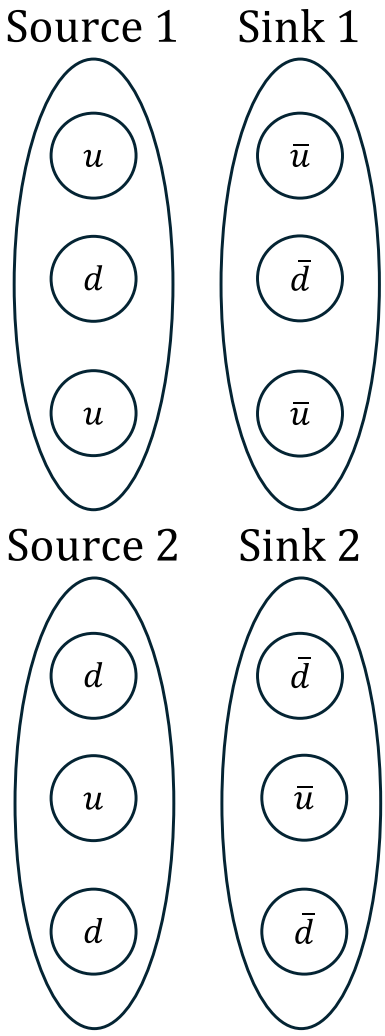


Figure 6.3: Dibaryon Dibaryon Physics Setup

6.2 Dibaryon-Dibaryon

6.2.1 Physics Setup

The physical situation of the dibaryon-dibaryon system features four lattice sites: two source nodes and two sink nodes. Each source node has three quarks with one node having two down and one up and the other having one down and two up. Each sink node has three quarks with one node having two down and one up and the other having one down and two up and all the quarks having antiness. These are represented in Figure 6.3.

6.2.2 Naive Code

The naive code generated features four nested loops over space (one for each lattice site), four nested weight loops, and two nested permutation loops over the symmetric group 3 for a total of ten main nested reduction domain loops. The summand consists of a product of six accesses to the propogator (which has six dimensions), four accesses to spacial weights, and six accesses to other weights.

6.2.3 Rewrites Applied

The initial naive IR can be seen at Listing A.2. Looking at the structure of the IR for dibaryon-dibaryon, one can notice that some of the permutation iterations may map to similar computations based on what spacial indices are used. This leads us to first expanding the up and down permutations so we have 36 groups. Then we reduce the space loops by partitioning each group. This leads to 4 groups with $O(N^2)$ scaling and 32 with $O(N^3)$ scaling which are both down from the original $O(N^4)$. The $O(N^2)$ groups represent when all the quarks from one source all map to the same sink while the $O(N^3)$ groups represent when the quarks from one source map to different sinks. We then group these into their two groups of asymptotic.

Listing 6.3 shows a snippet of the IR pertaining to the spacial loops for the case that becomes $O(N^3)$. In this snippet it is currently $O(N^4)$ where we have separated the $O(N^2)$ loops that we cannot partition on into a higher sum with the lower sum having the two spacial loops that can be partitioned. Listing 6.4 then shows the snippet once the spacial loops have been partitioned where you can see the asymptotic bound is now $O(N^3)$. The two precomputes have an isomorphic structure so they can have their expressions merged which we can see in Listing 6.5. The accesses on lines 14 and 15 show how the first access to the precomputed expression chooses which option to select. For the case which becomes $O(N^2)$ the IR looks like Listing 6.3 but the accesses added to the summand which have been

removed for brevity can be factored into two groups completely. This factorization can be seen in Listing 6.6 after the expression partitioning has been done. Then Listing 6.7 shows how the precomputes are able to merged as before.

```
summand_1 = 0.0
for src_p_src_6 in range(N):
    for src_src_4 in range(N):
        summand_2 = 0.0
        for snk_p_snk_10 in range(N):
            for snk_snk_8 in range(N):
                for w_snk_1_rank_62 in range(w_snk_1_rank):
                    for w_snk_2_rank_64 in range(w_snk_2_rank):
                        summand_2 += # REMOVED FOR BREVITY
        summand_1 += # REMOVED FOR BREVITY
output_328["REMOVED FOR BREVITY"] = summand_1
```

Listing 6.3: Dibaryon Dibaron Snippet for $O(N^3)$ before Expression Partitioning

```
summand_3 = 0.0
for src_p_src_6 in range(N):
    for src_src_4 in range(N):
        epsilon_9_332: NDArray = np.zeros(1,)
        summand_4 = 0.0
        for snk_p_snk_10 in range(N):
            for w_snk_2_rank_64 in range(w_snk_2_rank):
                summand_4 += # REMOVED FOR BREVITY
        epsilon_9_332[None] = summand_4
        epsilon_10_334: NDArray = np.zeros(1,)
        summand_5 = 0.0
        for snk_snk_8 in range(N):
            for w_snk_1_rank_62 in range(w_snk_1_rank):
                summand_5 += # REMOVED FOR BREVITY
        epsilon_10_334[None] = summand_5
        summand_3 += (psi1_92[srcSpaceRank_30, src_src_4, srcExternal_14]
```

```
    * \
                        psi2_94[srcSpaceRank_30, src_p_src_6, srcExternal_14
    ] * \
                          (1.0 * -1.0 * \
                           epsilon_9_332["REMOVED FOR BREVITY"] * \
                           epsilon_10_334["REMOVED FOR BREVITY"]))
output_336["REMOVED FOR BREVITY"] = summand_3
```

Listing 6.4: Dibaryon Dibaron Snippet for $O(N^3)$ after Expression Partitioning

```
summand_6 = 0.0
for src_p_src_6 in range(N):
    for src_src_4 in range(N):
        epsilon_9_11_342: NDArray = np.zeros(2)
        for choose_i_50_338 in range(2):
            summand_7 = 0.0
            for snk_p_snk_10 in range(N):
                for w_snk_2_rank_64 in range(w_snk_2_rank):
                    summand_7 += # REMOVED FOR BREVITY
            epsilon_9_11_342[choose_i_50_338] = summand_7
        summand_6 += (psi1_92[srcSpaceRank_30, src_src_4, srcExternal_14]
    * \
                        psi2_94[srcSpaceRank_30, src_p_src_6, srcExternal_14
    ] * \
                          (1.0 * -1.0 * \
                           epsilon_9_11_342[0, "REMOVED FOR BREVITY"] * \
                           epsilon_9_11_342[1, "REMOVED FOR BREVITY"]))
output_344["REMOVED FOR BREVITY"] = summand_6
```

Listing 6.5: Dibaryon Dibaron Snippet for $O(N^3)$ after Expression Partitioning and Merging

```
baryon_1_238: NDArray = np.zeros(1,)
summand_1 = 0.0
for snk_p_snk_10 in range(N):
    for src_p_src_6 in range(N):
```

```
        for w_snk_2_rank_64 in range(w_snk_2_rank):
            summand_1 += # REMOVED FOR BREVITY
baryon_1_238[None] = summand_1
baryon_2_240: NDArray = np.zeros(1,)
summand_2 = 0.0
for snk_snk_8 in range(N):
    for src_src_4 in range(N):
        for w_snk_1_rank_62 in range(w_snk_1_rank):
            summand_2 += # REMOVED FOR BREVITY
baryon_2_240[None] = summand_2
output_242["REMOVED FOR BREVITY"] = \
    (1.0 * 1.0 * baryon_1_238["REMOVED FOR BREVITY"] * \
                 baryon_2_240["REMOVED FOR BREVITY"])
```

Listing 6.6: Dibaryon Dibaron Snippet for $O(N^2)$ after Expression Partitioning

```
baryon_1_3_248: NDArray = np.zeros(2)
for choose_i_16_244 in range(2):
    summand_3 = 0.0
    for snk_p_snk_10 in range(N):
        for src_p_src_6 in range(N):
            for w_snk_2_rank_64 in range(w_snk_2_rank):
                summand_3 += # REMOVED FOR BREVITY
    baryon_1_3_248[choose_i_16_244] = summand_3
output_250["REMOVED FOR BREVITY"] = (1.0 * 1.0 * \
                    baryon_1_3_248[0, "REMOVED FOR BREVITY"] * \
                    baryon_1_3_248[1, "REMOVED FOR BREVITY"])
```

Listing 6.7: Dibaryon Dibaron Snippet for $O(N^2)$ after Expression Partitioning and Merging

We now focus on the $O(N^3)$ grouping as it dominates the runtime of the program. On this grouping we move the two spacial loops that we do not precompute over to the highest loop. Then we notice that many of the index choices with 32 options map to the same 2 options. This means that instead of doing a computation 32 times where we choose that same

thing several times we can instead compute two results and when doing the 32 iterations we can choose which result to pull from. We then apply some condense choice filter rewrites to achieve this goal. Finally we move index choices from inside the let to be instead in the use statement which can provide some flexibility when GPU scheduling. The complete rewrite code can be seen in Listing B.1 with the generated IR being at Listing A.3 (the precomputes have been removed from the printout for brevity).

6.2.4 Analysis of Rewrite Impact

Theoretical

The main result of our rewrites is the `partition_exprs` rewrite. This rewrite takes our spacial asymptotics from $O(N^4)$, where N is the size of the lattice, to $O(N^3)$ since we have removed a nested loop over space. The condense choice also adds constant factor scaling improvements by reducing the number of unique blocks that need to be precomputed.

Practical

Our results are summarized in Figure 6.4 and Figure 6.5. After we finished rewriting, we iteratively scheduled and profiled the code. We needed to tweak the rewrites four times as we sought to find versions that contained better schedules; this process was smooth and a good initial schedule was found in a few hours. The main time suck in scheduling was waiting for profiling data. Eventually, we found programs that outperformed the Tiramisu code on both V100s and A100s. We are still gathering data for A100s at present.

On smaller sizes, the pre-computations of access yield a non-trivial constant factor, leading to speeds up of 5x to 10x. On larger sizes, the asymptotic costs dominate so we get closer to the base Tiramisu code. However, our rewrites to move around sums and access as well as condensing duplicates choices and creating let bindings allowed to us find a version that outperformed the Tiramisu by a consistent 1.25x on the larger sizes on the V100. We believe

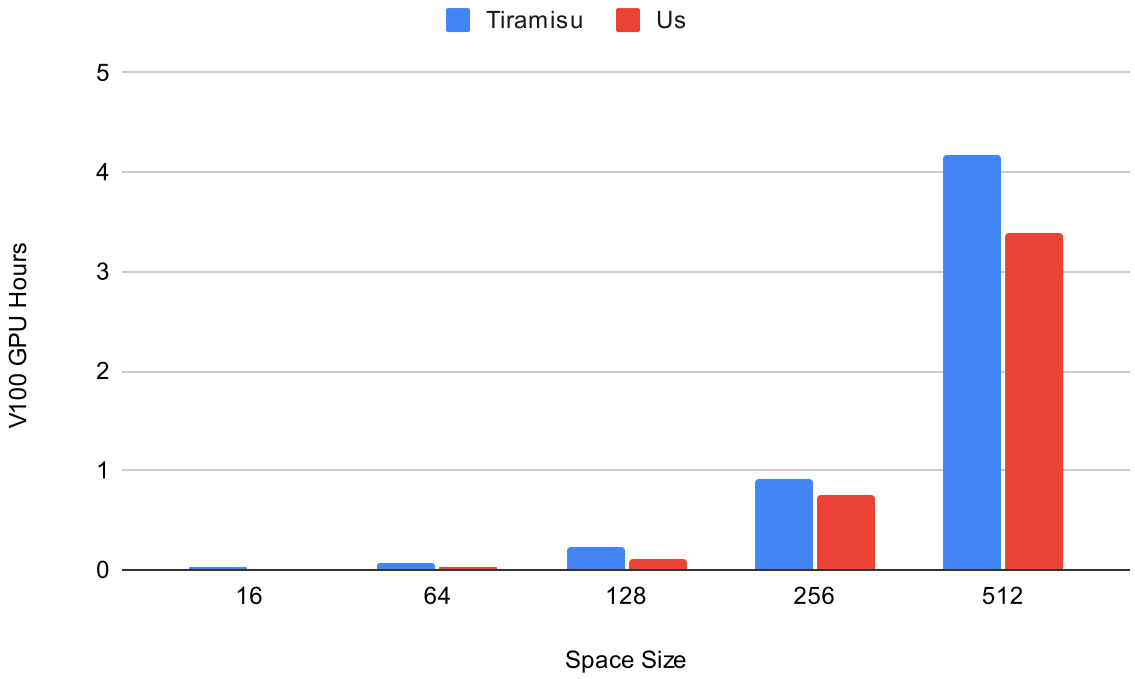


Figure 6.4: Timings for a scheduled Dibaryon-Dibaryon on different space sizes for a V100

that we can still do better. Most of the rewrites got us most of the way there by hitting the asymptotic costs, but they also helped us deal with the constant factors associated with complex computer architectures.

6.2.5 Impact

The Dibaryon Dibaryon case in particular has significance to the physicists because of its large scaling compared to the other problems described here. The ability of our system to quickly generate asymptotically optimal code has allowed a much better development time and allowed more time devoted to targeting machine architecture instead of on algorithmic correctness. For reference, our system takes around 100 SLOC to define the system, around 100 to perform the rewrites, and 100-200 additional scheduling lines in the Halide file. This is in contrast to the 1000s of SLOC required to do the computations in the previous system which also must have their correctness reasoned as a whole entity instead of people able to be

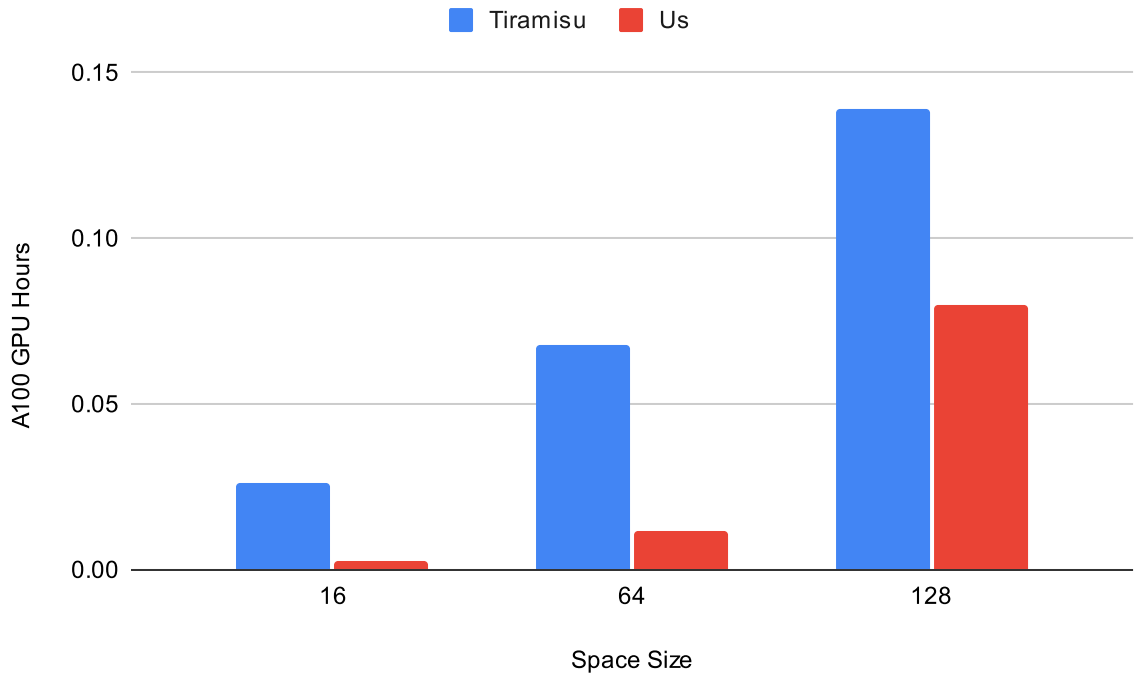


Figure 6.5: Timings for a scheduled Dibaryon-Dibaryon on different space sizes for a A100

reasoned about in parts one can with the new system. The comparison of SLOC between our system and the physicist's Tiramisu code can be seen for all the case studies in Figure 6.9.

6.3 Dibaryon-Hexaquark

6.3.1 Physics Setup

The physical situation of the dibaryon-hexaquark system features three lattice sites: two source nodes and one sink node. Each source node has three quarks with one node having two down and one up and the other having one down and two up. The sink node has six quarks with three up and three down and all the quarks having antiness. These are represented in Figure 6.6.

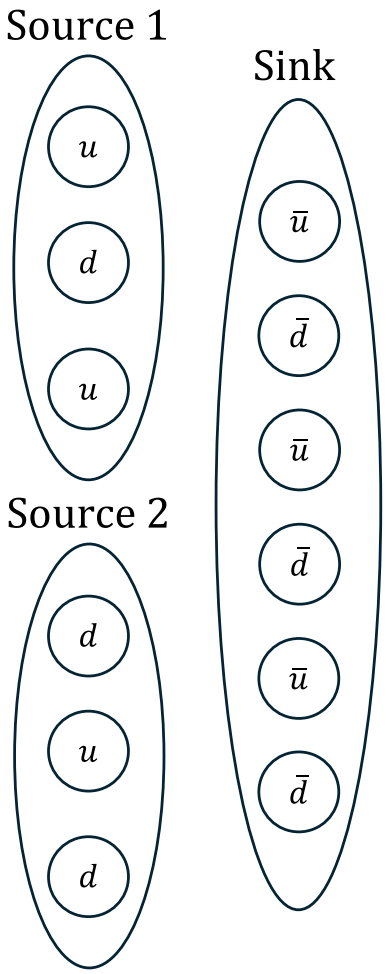


Figure 6.6: Dibaryon Hexaquark Physics Setup

6.3.2 Naive Code

The naive code generated features three nested loops over space (one for each lattice site), three nested weight loops, and two nested permutation loops over the symmetric group 3 for a total of eight main nested reduction domain loops. The summand consists of a product of six accesses to the propogator (which has six dimensions), three accesses to spacial weights, and four accesses to other weights.

6.3.3 Rewrites Applied

The initial naive IR can be seen in Listing A.4. To begin, we first apply `simplify_conj`, which moves a conj of an `Add`/`Mult` into its children instead. This allows separating spacial accesses later since they are no longer combined with other parts under a `Conj`. Then we apply `remove_unnecessary_index_choice` which removes `IndexChoice`es which have all choices that are the same, which is the case for the second spacial access of all the propagators.

At this point we notice that all the propagators depend on the sink but they are split into groups of three based on which source they depend on (and this also matches what weight index they depend on). This means we want to apply a partition expression call at some point. To apply this call we need the iteration indices to not have parts of the summand that share all of them so we need to use `separate_sum` to move some of the iteration indices to a separate sum. We apply `separate_sum` where we move the variables and we specify the indices to move as all the indices besides the sink and weight indices that will split the propagators into groups of three.

Now that we have the inner sum we can apply `partition_exprs` to it. Once the partition has been done, we can merge the lets of the multilet generated since the let expressions are isomorphic. The final result of the IR can be seen in Listing A.5.

```
1 def dib_hex_rewrite(comp):
```

```
    res = remove_unnecessary_index_choice(simplify_conj(comp))
    res = separate_sum(res, ['w_snk_H_rank', 'up', 'down', 'snkSpaceRank',
    'srcSpaceRank', 'srcSigma1', 'srcSigma2', 'snk_snk'], move_vars=True)

    inner_sum_key = [2, 2, 0, 3]
    res = run_on_loc(res, inner_sum_key, partition_exprs)
    res = run_on_loc(res, inner_sum_key, merge_multilet)
    return res
```

Listing 6.8: Dibaryon Hexaquark Rewrites

6.3.4 Analysis of Rewrite Impact

Theoretical

The main result of our rewrites is the `partition_exprs` rewrite. This rewrite takes our inner loop asymptotics from $O(N^3W^3)$, where N is the size of the lattice and W is the number of weights, to $O(N^2W^2)$ since we have removed a nested loop of both space and weights. All the other loops remain the same because there is no way to avoid iterating over them.

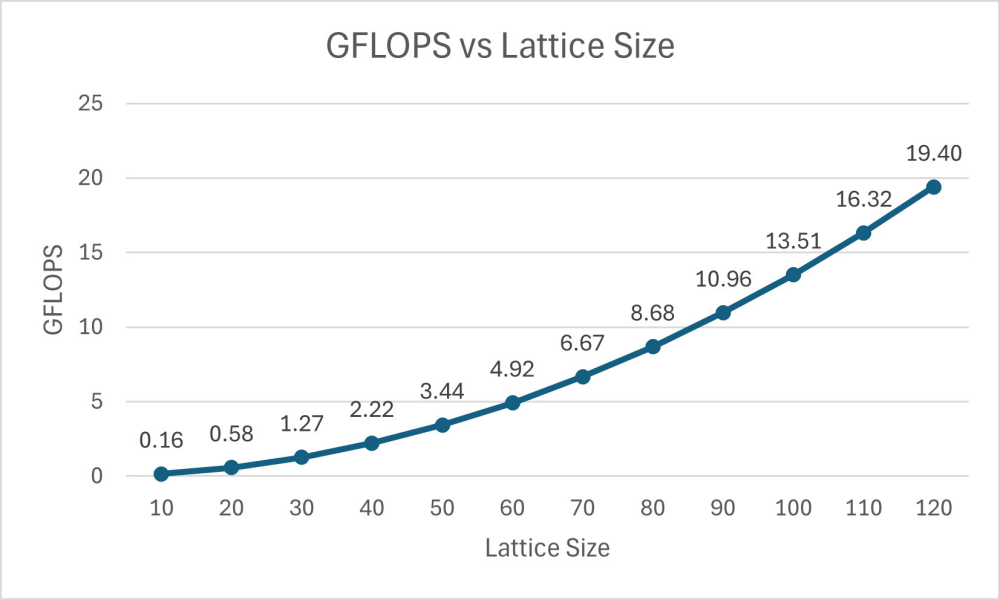
Practical

As shown in Figure 6.7, the rewritten code scales with N^2 where N is the lattice size. The unrewritten code takes long enough that a timing comparison is not included because of the time needed to generate data.

6.4 Hexaquark-Hexaquark

6.4.1 Physics Setup

This physical situation of the hexaquark-hexaquark system features two lattice sites: one source node and one sink node. The source node has six quarks with three up and three



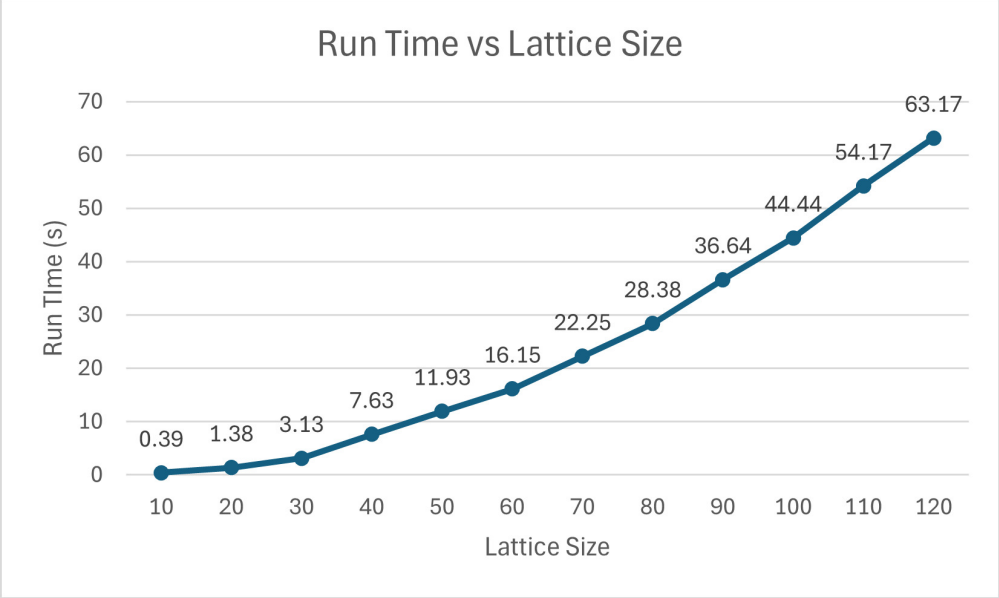


Figure 6.7: Graphs showing how the GFLOPS and runtime change as the lattice size (N) increases for the Dibaryon Hexaquark case study.

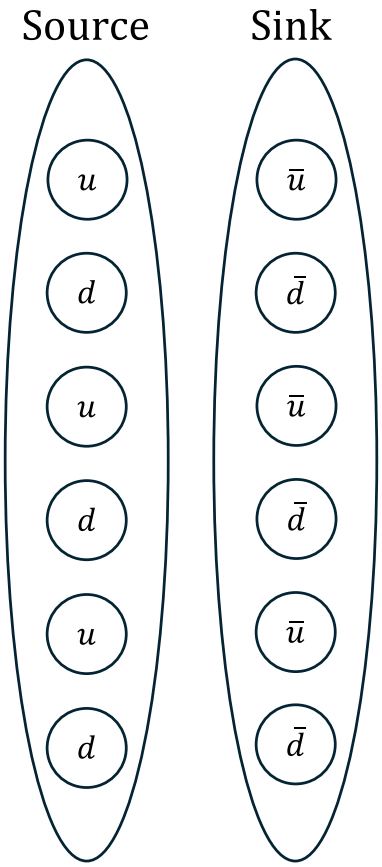


Figure 6.8: Hexaquark Hexaquark Physics Setup

down. The sink node has six quarks with three up and three down and all the quarks having antiness. These are represented in Figure 6.8.

6.4.2 Naive Code

The naive code generated features two nested loops over space (one for each lattice site), two nested weight loops, and two nested permutation loops over the symmetric group 3 for a total of siz main nested reduction domain loops. The summand consists of a product of six accesses to the propogator (which has six dimensions), two accesses to spacial weights, and two accesses to other weights.

6.4.3 Rewrites Applied

We first apply `remove_unnecessary_index_choice` which removes `IndexChoice`es which have all choices that are the same, which is the case for the second spacial access of all the propagators. This cleans up the IR and allows us to realize that there are no asymptotic optimizations possible. There are no optimizations because we have already achieved the minimum number of nested loops needed in order to compute the mathematical structure since all the propagators depend on all the indices we loop over. The unchanged IR and the changed IR can be seen at Listing A.6 and Listing A.7 respectively.

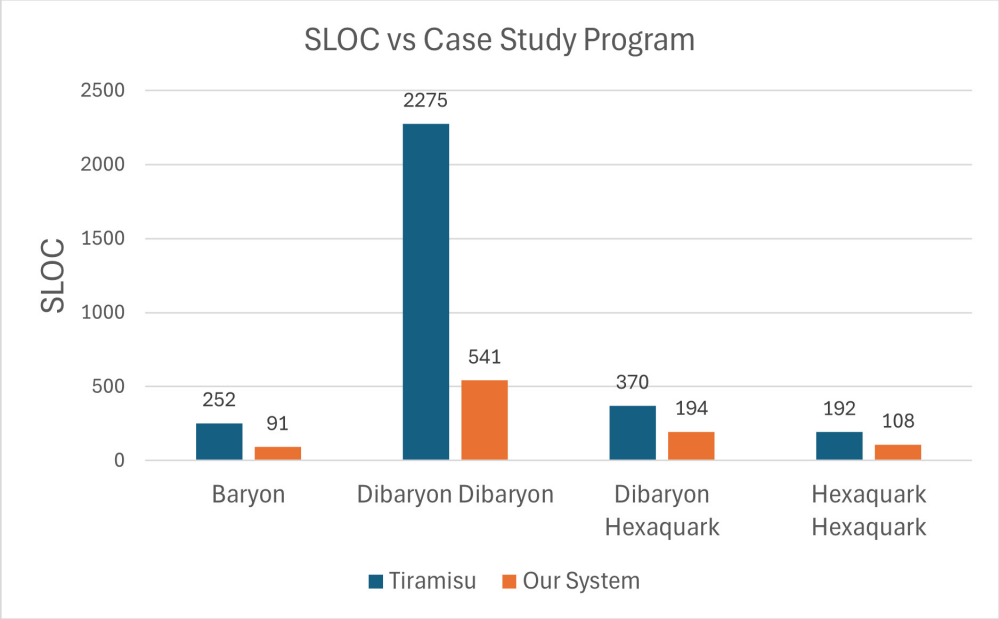


Figure 6.9: Comparision of SLOC for the physicist's Tiramisu code versus our system. The Dibaryon Dibaryon case also includes the SLOC for our GPU scheduling (the other cases were not GPU scheduled).

Chapter 7

Future Work

7.1 Automatic Algorithmic Optimization

The most promising area to speed up the development process for solving these computations is automatic optimizations. With manual optimizations, one must first look at the generated IR and then use the optimization functions with specific inputs to rewrite the IR into the desired optimal structure. An automatic system for applying optimizations would allow the physicists to simply specify their desired problem and be able to immediately obtain an optimized program for getting their results. Finding an efficient way for finding optimal rewrites can pose a challenge because the space of potential rewrites can be large or lead to extraneous rewrites.

One potential way to approach rewrites is to make a pipeline of applying rewrites. The algorithm could first attempt to apply algorithmic optimizations, then search for duplicate structures and merge any `MultiLet`s that are eligible. Afterwards, it could attempt expanding a permutation followed by constant propagation. It could then repeat these steps until some decided ending criteria is met.

Another approach could follow more of a search method. It could take a given structure and try different possible rewrites on it and use some heuristics for evaluating the asymptotics

and FLOPs of a given structure so we could perform an A* search. The search could continue till we meet some end condition, most likely based on the heuristic of the leaf reaching a certain threshold.

To complement the other methods, there are specific optimizations one could attempt to apply based on the IR value. Given a `Sum` over a permutation one could immediately choose to expand it. Another is always attempting to propagate constants after every other optimization attempt. Also, precomputation can be almost always left till the end because it increases the complexity of the structure which can make it harder to optimize a structure after applying it. The FLOP savings of precomputation should be the same no matter when we apply the optimization because any structure we could save time by precomputing will remain after other optimizations are done because if it is gone then it was optimized to a better structure.

7.2 Automatic GPU Scheduling

Once the Halide generation file has been created from the IR, one has to manually schedule it to get competetive results especially for use on GPUs. The process of manually scheduling can be tedious and is a prime candidate for automatic generation since a schedule could potentially be inserted when creating the Halide generation file.

A flag for enabling GPU scheduling could be added to the Halide generation process and parameters be added to describe the machine that will be running the program. The automatic scheduler could use the parameters to create optimal cache usage and use the IR structure to determine an optimal scheduling from a set of schedules that have been manually scheduled before for similar structures.

Chapter 8

Conclusion

Based on the flexibility of the rewrite rules and ability to represent desired computations in our IR, the LQCD IR DSL we have created is a powerful tool to speed up the development of solving new Lattice Quantum Chromodynamics problems. It is able to easily generate naive computations given the physics problem description so that initial benchmarks can be found. From there, testing various algorithms can be done quickly by applying different rewrites with minimal code needed. This flexibility for changing algorithms quickly with correctness greatly improves the work flow of previous methods which required tremendous amounts of handwritten code for each algorithm one desired to test.

The case studies show that a variety of scenarios can be easily generated with our system. Using the visual outputs of the IR also allows easy inspection of the formula involved which can make finding optimizations smoother. Without an IR, one would need a complete mental model of their code and how it represents the manually rewritten formulas in code. The ability to see the progress on changing the formula visually can speed up development greatly.

The quicker development time to GPU scheduling also greatly removes a bottleneck for optimizing new problems. In the past, one needed to spend large amounts of time forming the overall code with correct asymptotics before attempting to schedule it on a GPU or

had to intermix the scheduling with the algorithm development. Either of these approaches form a large bottleneck before being able to spend time fine tuning the program for the targeted machine. With the IR and rewrite system, one can quickly apply rewrites and confirm astmptotics of the program which then allows moving onto the scheduling process quickly. Overall, this work marks a significant step forward in being able to solve larger LQCD problems through much better development time and code correctness.

Appendix A

Large LQCD IR Printouts

A.1 Baryon IR

```
output_348: NDArray = np.zeros(1, )
summand_6 = 0.0
for s_snk_298 in range(N):
    for s_src_296 in range(N):
        let_strngth_2_346: NDArray = np.zeros(2, 3, 3, 2, 2, 3)
        for i_2_12_334 in range(2):
            for i_3_13_336 in range(3):
                for i_3_14_338 in range(3):
                    for i_2_15_340 in range(2):
                        for i_2_16_342 in range(2):
                            for i_3_17_344 in range(3):
                                summand_7 = 0.0
                                for w_src_1_rank_314 in range(w_src_1_rank):
                                    summand_7 += (
                                        w_src_322[w_src_1_rank_314] * S_318[
                                            s_src_296,
                                            w_src_spin_0_300[w_src_1_rank_314],
                                            w_src_color_0_300[w_src_1_rank_314],
                                            s_snk_298, i_2_12_334, i_3_13_336]
                                        * S_318[
                                            s_src_296,
                                            w_src_spin_2_304[w_src_1_rank_314],
                                            w_src_color_2_304[w_src_1_rank_314],
```

```
                                        s_snk_298, i_2_16_342, i_3_14_338]
                                    * S_318[
                                        s_src_296,
                                        w_src_spin_1_302[w_src_1_rank_314],
                                        w_src_color_1_302[w_src_1_rank_314],
                                        s_snk_298, i_2_15_340, i_3_17_344]
                                    * 1.0)
                                let_strngth_2_346[i_2_12_334, i_3_13_336,
                                                  i_3_14_338, i_2_15_340,
                                                  i_2_16_342,
                                                  i_3_17_344] = summand_7
        summand_8 = 0.0
        for w_src_2_rank_316 in range(w_src_1_rank):
            for up_328 in itertools.permutations(range(2)):
                summand_8 += (
                    sign(up_328) * w_snk_326[w_src_2_rank_316] *
                    let_strngth_2_346[s_snk_298, s_src_296, [
                        w_snk_spin_0_306[w_src_2_rank_316],
                        w_snk_spin_2_310[w_src_2_rank_316]
                    ][up_328[0]], [
                        w_snk_color_0_306[w_src_2_rank_316],
                        w_snk_color_2_310[w_src_2_rank_316]
                    ][up_328[0]], [
                        w_snk_color_0_306[w_src_2_rank_316],
                        w_snk_color_2_310[w_src_2_rank_316]
                    ][up_328[1]], w_snk_spin_1_308[w_src_2_rank_316], [
                        w_snk_spin_0_306[w_src_2_rank_316],
                        w_snk_spin_2_310[w_src_2_rank_316]
                    ][up_328[1]], w_snk_color_1_308[w_src_2_rank_316]])
        summand_6 += (psi_320[s_src_296] * np.conj(phi_324[s_snk_298]) *
                      summand_8)
output_348[None] = summand_6
```

Listing A.1: Baryon Rewritten IR

A.2 Dibaryon Dibaryon IR

```
output_2078: NDArray = np.zeros(EN, rhoSnkSize, EN, rhoSrcSize)
for snkExternal_16 in range(EN):
```

```
    for rhoSnk_24 in range(rhoSnkSize):
        for srcExternal_14 in range(EN):
            for rhoSrc_22 in range(rhoSrcSize):
                summand_0 = 0.0
                for srcSigma1_42 in range(srcSigma1):
                    for snk_p_snk_10 in range(N):
                        for srcSpaceRank_30 in range(srcSpaceRank):
                            for src_p_src_6 in range(N):
                                for w_src_1_rank_58 in range(w_src_1_rank):
                                    for w_src_2_rank_60 in range(w_src_2_rank):
                                        for w_snk_2_rank_64 in range(w_snk_2_rank):
                                            for srcSigma2_44 in range(srcSigma2):
                                                for snkSigma2_48 in range(snkSigma2):
                                                    for w_snk_1_rank_62 in range(
w_snk_1_rank):
                                                        for src_src_4 in range(N):
                                                            for snkSigma1_46 in range(
snkSigma1):
                                                                for snk_snk_8 in range(N):
                                                                    for snkSpaceRank_32 in
range(snkSpaceRank):
                                                                        for up_112 in
itertools.permutations(range(3)):
                                                                            for down_114 in
itertools.permutations(range(3)):
                                                                                summand_0 +=
 (S_90[src_src_4, w_src_1_spin_0_66[w_src_1_rank_58, srcSigma1_42], w_src_1_color_0_66[
w_src_1_rank_58, srcSigma1_42], [snk_snk_8, snk_snk_8, snk_p_snk_10][up_112[0]], [
w_snk_1_spin_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_spin_2_82[w_snk_1_rank_62,
snkSigma1_46], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[0]], [
w_snk_1_color_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_color_2_82[w_snk_1_rank_62,
snkSigma1_46], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[0]]] * S_90[
src_src_4, w_src_1_spin_2_70[w_src_1_rank_58, srcSigma1_42], w_src_1_color_2_70[
w_src_1_rank_58, srcSigma1_42], [snk_snk_8, snk_snk_8, snk_p_snk_10][up_112[1]], [
w_snk_1_spin_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_spin_2_82[w_snk_1_rank_62,
snkSigma1_46], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[1]], [
w_snk_1_color_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_color_2_82[w_snk_1_rank_62,
snkSigma1_46], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[1]]] * S_90[
src_p_src_6, w_src_2_spin_1_74[w_src_2_rank_60, srcSigma2_44], w_src_2_color_1_74[
w_src_2_rank_60, srcSigma2_44], [snk_snk_8, snk_snk_8, snk_p_snk_10][up_112[2]], [
w_snk_1_spin_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_spin_2_82[w_snk_1_rank_62,
```

```
    snkSigma1_46], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[2]], [
    w_snk_1_color_0_78[w_snk_1_rank_62, snkSigma1_46], w_snk_1_color_2_82[w_snk_1_rank_62,
    snkSigma1_46], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48]][up_112[2]]] * sign(
    up_112) * S_90[src_src_4, w_src_1_spin_1_68[w_src_1_rank_58, srcSigma1_42],
    w_src_1_color_1_68[w_src_1_rank_58, srcSigma1_42], [snk_snk_8, snk_p_snk_10,
    snk_p_snk_10][down_114[0]], [w_snk_1_spin_1_80[w_snk_1_rank_62, snkSigma1_46],
    w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64,
    snkSigma2_48]][down_114[0]], [w_snk_1_color_1_80[w_snk_1_rank_62, snkSigma1_46],
    w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64,
    snkSigma2_48]][down_114[0]]] * S_90[src_p_src_6, w_src_2_spin_0_72[w_src_2_rank_60,
    srcSigma2_44], w_src_2_color_0_72[w_src_2_rank_60, srcSigma2_44], [snk_snk_8,
    snk_p_snk_10, snk_p_snk_10][down_114[1]], [w_snk_1_spin_1_80[w_snk_1_rank_62,
    snkSigma1_46], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[
    w_snk_2_rank_64, snkSigma2_48]][down_114[1]], [w_snk_1_color_1_80[w_snk_1_rank_62,
    snkSigma1_46], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[
    w_snk_2_rank_64, snkSigma2_48]][down_114[1]]] * S_90[src_p_src_6, w_src_2_spin_2_76[
    w_src_2_rank_60, srcSigma2_44], w_src_2_color_2_76[w_src_2_rank_60, srcSigma2_44], [
    snk_snk_8, snk_p_snk_10, snk_p_snk_10][down_114[2]], [w_snk_1_spin_1_80[w_snk_1_rank_62,
     snkSigma1_46], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[
    w_snk_2_rank_64, snkSigma2_48]][down_114[2]], [w_snk_1_color_1_80[w_snk_1_rank_62,
    snkSigma1_46], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[
    w_snk_2_rank_64, snkSigma2_48]][down_114[2]]] * sign(down_114) * (psi1_92[
    srcSpaceRank_30, src_src_4, srcExternal_14] * psi2_94[srcSpaceRank_30, src_p_src_6,
    srcExternal_14] * w_src_1_96[w_src_1_rank_58, srcSigma1_42] * w_src_2_98[w_src_2_rank_60
    , srcSigma2_44] * v_src_100[srcSigma1_42, srcSigma2_44, rhoSrc_22]) * np.conj((phi1_102[
    snkSpaceRank_32, snk_snk_8, snkExternal_16] * phi2_104[snkSpaceRank_32, snk_p_snk_10,
    snkExternal_16] * w_snk_1_106[w_snk_1_rank_62, snkSigma1_46] * w_snk_2_108[
    w_snk_2_rank_64, snkSigma2_48] * v_snk_110[snkSigma1_46, snkSigma2_48, rhoSnk_24])))
24                output_2078[snkExternal_16, rhoSnk_24, srcExternal_14, rhoSrc_22] =
    summand_0
```

Listing A.2: Dibaryon Dibaryon Naive IR

```
1 output_2080: NDArray = np.zeros(EN, rhoSnkSize, EN, rhoSrcSize)
2 for snkExternal_16 in range(EN):
3     for rhoSnk_24 in range(rhoSnkSize):
4         for srcExternal_14 in range(EN):
5             for rhoSrc_22 in range(rhoSrcSize):
6                 sep_unroll_perm_2_3510658113350937887_epsilon_144_sumed_1958: NDArray = np.
    zeros(1,)
7                 summand_1 = 0.0
8                 for srcSpaceRank_30 in range(srcSpaceRank):
```

```
                    for src_p_src_6 in range(N):
                        for src_src_4 in range(N):
                            summand_2 = 0.0
                            for snkSpaceRank_32 in range(snkSpaceRank):
                                for i_condensed_614_1944 in range(32):
                                    for srcSigma1_42 in range(srcSigma1):
                                        for srcSigma2_44 in range(srcSigma2):
                                            for w_src_1_rank_58 in range(w_src_1_rank):
                                                for w_src_2_rank_60 in range(w_src_2_rank):
                                                    for snkSigma1_46 in range(snkSigma1):
                                                        for snkSigma2_48 in range(snkSigma2)
:
                                                            acc_pre_2024: NDArray = np.zeros
(1,)
                                                            # PRECOMPUTES REMOVED FOR
BREVITY
                                                            acc_pre_2052: NDArray = np.zeros
(1,)
                                                            acc_pre_2052[None] = [0, 0, 0,
0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 5, 5, 5, 5, 1, 1, 2, 2, 2, 2, 4, 4, 5, 5, 5, 5][
i_condensed_614_1944]
                                                            merged_eps_0_2016: NDArray = np.
zeros(2, 3, 2, 3, 2, 3, 6, 2)
                                                            for sc_2_1966 in range(2):
                                                                for sc_3_1968 in range(3):
                                                                    for sc_2_1970 in range
(2):
                                                                        for sc_3_1972 in
range(3):
                                                                            for sc_2_1974 in
 range(2):
                                                                                for
sc_3_1976 in range(3):
                                                                                    for
i_6_2012 in range(6):
                                                                                        for
i_2_2014 in range(2):

merged_eps_0_compressed_2010: NDArray = np.zeros(6, 2)

for i_6_2006 in range(6):
```

```
    for choose_eps_612_compressed_2008 in range(2):
        merged_eps_0_compressed_compressed_1994: NDArray = np.zeros(6, 2)
        for choose_eps_612_compressed_1992 in range(6):
            for choose_eps_612_compressed_1984 in range(2):
                summand_3 = 0.0
                for snk_p_snk_10 in range(N):
                    summand_4 = 0.0
                    for w_snk_2_rank_64 in range(w_snk_2_rank):
                        acc_pre_2054: NDArray = np.zeros(1,)
                        acc_pre_2054[None] = [w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1992]
                        acc_pre_2056: NDArray = np.zeros(1,)
                        acc_pre_2056[None] = [w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1992]
                        acc_pre_2058: NDArray = np.zeros(1,)
                        acc_pre_2058[None] = [w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1992]
```

```
                        acc_pre_2060: NDArray = np.zeros(1,)

                        acc_pre_2060[None] = [w_snk_2_color_1_86[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_1_86[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48],
w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64,
snkSigma2_48]][choose_eps_612_compressed_1992]

                        acc_pre_2062: NDArray = np.zeros(1,)

                        acc_pre_2062[None] = [w_snk_2_spin_2_88[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48],
w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64,
snkSigma2_48]][choose_eps_612_compressed_1992]

                        acc_pre_2064: NDArray = np.zeros(1,)

                        acc_pre_2064[None] = [w_snk_2_color_2_88[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_0_84[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48],
w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64,
snkSigma2_48]][choose_eps_612_compressed_1992]

                        summand_4 += (S_90[src_src_4, sc_2_1966, sc_3_1968, snk_p_snk_10
, acc_pre_2054[w_snk_2_rank_64, snkSigma2_48, choose_eps_612_compressed_1992],
acc_pre_2056[w_snk_2_rank_64, snkSigma2_48, choose_eps_612_compressed_1992]] * S_90[[
src_p_src_6, src_src_4][choose_eps_612_compressed_1984], sc_2_1970, sc_3_1972,
snk_p_snk_10, acc_pre_2058[w_snk_2_rank_64, snkSigma2_48, choose_eps_612_compressed_1992
], acc_pre_2060[w_snk_2_rank_64, snkSigma2_48, choose_eps_612_compressed_1992]] * S_90[
src_p_src_6, sc_2_1974, sc_3_1976, snk_p_snk_10, acc_pre_2062[w_snk_2_rank_64,
snkSigma2_48, choose_eps_612_compressed_1992], acc_pre_2064[w_snk_2_rank_64,
snkSigma2_48, choose_eps_612_compressed_1992]] * w_snk_2_108[w_snk_2_rank_64,
snkSigma2_48])

                    summand_3 += (np.conj(phi2_104[snkSpaceRank_32, snk_p_snk_10,
snkExternal_16]) * summand_4)

                merged_eps_0_compressed_compressed_1994[choose_eps_612_compressed_1992,
choose_eps_612_compressed_1984] = summand_3

```

```
        merged_eps_0_compressed_2010[i_6_2006, choose_eps_612_compressed_2008] =
merged_eps_0_compressed_compressed_1994[i_6_2006, snkExternal_16, snkSigma2_48,
snkSpaceRank_32, src_p_src_6, src_src_4, sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972,
sc_2_1974, sc_3_1976, choose_eps_612_compressed_2008]

merged_eps_0_2016[sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972, sc_2_1974, sc_3_1976,
i_6_2012, i_2_2014] = merged_eps_0_compressed_2010[snkExternal_16, snkSigma2_48,
snkSpaceRank_32, src_p_src_6, src_src_4, sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972,
sc_2_1974, sc_3_1976, i_6_2012, i_2_2014]
                                                    merged_eps_1_2022: NDArray = np.
zeros(2, 3, 2, 3, 2, 3, 6, 2)
                                                    for sc_2_1966 in range(2):
                                                        for sc_3_1968 in range(3):
                                                            for sc_2_1970 in range
(2):
                                                                for sc_3_1972 in
range(3):
                                                                    for sc_2_1974 in
 range(2):
                                                                        for
sc_3_1976 in range(3):
                                                                            for
i_6_2018 in range(6):
                                                                                for
i_2_2020 in range(2):

merged_eps_1_compressed_2004: NDArray = np.zeros(6, 2)

for i_6_2000 in range(6):

    for choose_eps_612_compressed_2002 in range(2):

        merged_eps_1_compressed_compressed_1998: NDArray = np.zeros(6, 2)

        for choose_eps_612_compressed_1996 in range(6):

            for choose_eps_612_compressed_1988 in range(2):

                summand_5 = 0.0

                for snk_p_snk_10 in range(N):
```

```
                    summand_6 = 0.0

                    for w_snk_2_rank_64 in range(w_snk_2_rank):

                        acc_pre_2066: NDArray = np.zeros(1,)

                        acc_pre_2066[None] = [w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1996]

                        acc_pre_2068: NDArray = np.zeros(1,)

                        acc_pre_2068[None] = [w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1996]

                        acc_pre_2070: NDArray = np.zeros(1,)

                        acc_pre_2070[None] = [w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1996]

                        acc_pre_2072: NDArray = np.zeros(1,)

                        acc_pre_2072[None] = [w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48]][choose_eps_612_compressed_1996]

                        acc_pre_2074: NDArray = np.zeros(1,)

                        acc_pre_2074[None] = [w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[
```

```
    w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48],
    w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64,
    snkSigma2_48]][choose_eps_612_compressed_1996]
89
                            acc_pre_2076: NDArray = np.zeros(1,)
90
                            acc_pre_2076[None] = [w_snk_1_color_2_82[w_snk_2_rank_64,
    snkSigma2_48], w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[
    w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48],
    w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64,
    snkSigma2_48]][choose_eps_612_compressed_1996]
91
                            summand_6 += (S_90[src_src_4, sc_2_1966, sc_3_1968, snk_p_snk_10
    , acc_pre_2066[choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48],
    acc_pre_2068[choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48]] * S_90[[
    src_p_src_6, src_src_4][choose_eps_612_compressed_1988], sc_2_1970, sc_3_1972,
    snk_p_snk_10, acc_pre_2070[choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48
    ], acc_pre_2072[choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48]] * S_90[
    src_p_src_6, sc_2_1974, sc_3_1976, snk_p_snk_10, acc_pre_2074[
    choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48], acc_pre_2076[
    choose_eps_612_compressed_1996, w_snk_2_rank_64, snkSigma2_48]] * w_snk_1_106[
    w_snk_2_rank_64, snkSigma2_48])
92
                        summand_5 += (np.conj(phi1_102[snkSpaceRank_32, snk_p_snk_10,
    snkExternal_16]) * summand_6)
93
                    merged_eps_1_compressed_compressed_1998[choose_eps_612_compressed_1996,
    choose_eps_612_compressed_1988] = summand_5
94
            merged_eps_1_compressed_2004[i_6_2000, choose_eps_612_compressed_2002] =
    merged_eps_1_compressed_compressed_1998[i_6_2000, snkExternal_16, snkSigma2_48,
    snkSpaceRank_32, src_p_src_6, src_src_4, sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972,
    sc_2_1974, sc_3_1976, choose_eps_612_compressed_2002]
95
    merged_eps_1_2022[sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972, sc_2_1974, sc_3_1976,
    i_6_2018, i_2_2020] = merged_eps_1_compressed_2004[snkExternal_16, snkSigma2_48,
    snkSpaceRank_32, src_p_src_6, src_src_4, sc_2_1966, sc_3_1968, sc_2_1970, sc_3_1972,
    sc_2_1974, sc_3_1976, i_6_2018, i_2_2020]
96                                                      merged_vars_epsilon_9_11_1962:
    NDArray = np.zeros(1,)
97                                                      merged_vars_epsilon_9_11_1962[
```

```
    None] = merged_eps_0_2016[snkExternal_16, snkSigma2_48, snkSpaceRank_32, src_p_src_6,
    src_src_4, acc_pre_2024[srcSigma1_42, srcSigma2_44, i_condensed_614_1944,
    w_src_2_rank_60, w_src_1_rank_58], acc_pre_2026[srcSigma1_42, srcSigma2_44,
    i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58], acc_pre_2028[srcSigma1_42,
    srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58], acc_pre_2030[
    srcSigma1_42, srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58],
    acc_pre_2032[srcSigma1_42, srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60,
    w_src_1_rank_58], acc_pre_2034[srcSigma1_42, srcSigma2_44, i_condensed_614_1944,
    w_src_2_rank_60, w_src_1_rank_58], acc_pre_2036[i_condensed_614_1944], acc_pre_2038[
    i_condensed_614_1944]]
98                                                      merged_vars_epsilon_9_11_1964:
    NDArray = np.zeros(1,)
99                                                      merged_vars_epsilon_9_11_1964[
    None] = merged_eps_1_2022[snkExternal_16, snkSigma1_46, snkSpaceRank_32, src_src_4,
    src_p_src_6, acc_pre_2040[srcSigma1_42, srcSigma2_44, i_condensed_614_1944,
    w_src_2_rank_60, w_src_1_rank_58], acc_pre_2042[srcSigma1_42, srcSigma2_44,
    i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58], acc_pre_2044[srcSigma1_42,
    srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58], acc_pre_2046[
    srcSigma1_42, srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60, w_src_1_rank_58],
    acc_pre_2048[srcSigma1_42, srcSigma2_44, i_condensed_614_1944, w_src_2_rank_60,
    w_src_1_rank_58], acc_pre_2050[srcSigma1_42, srcSigma2_44, i_condensed_614_1944,
    w_src_2_rank_60, w_src_1_rank_58], acc_pre_2052[i_condensed_614_1944], acc_pre_2038[
    i_condensed_614_1944]]
100                                                     summand_2 += (w_src_1_96[
    w_src_1_rank_58, srcSigma1_42] * w_src_2_98[w_src_2_rank_60, srcSigma2_44] * v_src_100[
    srcSigma1_42, srcSigma2_44, rhoSrc_22] * v_snk_110[snkSigma1_46, snkSigma2_48, rhoSnk_24
    ] * ([1.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0,
    1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0][
    i_condensed_614_1944] * [-1.0, 1.0, 1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0, -1.0,
     -1.0, -1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0, 1.0, -1.0, -1.0, 1.0, 1.0, -1.0,
    -1.0, -1.0, -1.0, -1.0, -1.0, -1.0][i_condensed_614_1944] *
    merged_vars_epsilon_9_11_1962[srcSigma1_42, src_p_src_6, w_src_1_rank_58, src_src_4,
    srcSigma2_44, i_condensed_614_1944, snkExternal_16, snkSigma2_48, w_src_2_rank_60,
    snkSpaceRank_32] * merged_vars_epsilon_9_11_1964[srcSigma1_42, src_p_src_6,
    w_src_1_rank_58, src_src_4, srcSigma2_44, i_condensed_614_1944, snkExternal_16,
    w_src_2_rank_60, snkSigma1_46, snkSpaceRank_32]))
101                         summand_1 += (psi1_92[srcSpaceRank_30, src_src_4, srcExternal_14
    ] * psi2_94[srcSpaceRank_30, src_p_src_6, srcExternal_14] * summand_2)
102             sep_unroll_perm_2_3510658113350937887_epsilon_144_sumed_1958[None] =
    summand_1
103             sep_unroll_perm_0_7330295488186759212_baryon_145_sumed_1960: NDArray = np.
```

```
    zeros(1,)
            merged_vars_baryon_1_3_1948: NDArray = np.zeros(4, 2, 2, 2, 2, 3, 3, 3,
snkSigma2, snkSpaceRank, srcSpaceRank, N)
            for choose_eps_613_1930 in range(4):
                for choose_i_16_240 in range(2):
                    for i_2_12_224 in range(2):
                        for i_2_2_204 in range(2):
                            for i_2_7_214 in range(2):
                                for i_3_11_222 in range(3):
                                    for i_3_5_210 in range(3):
                                        for i_3_8_216 in range(3):
                                            for snkSigma2_48 in range(snkSigma2):
                                                for snkSpaceRank_32 in range(
snkSpaceRank):
                                                    for srcSpaceRank_30 in range(
srcSpaceRank):
                                                        for src_p_src_6 in range(N):
                                                            summand_7 = 0.0
                                                            for snk_p_snk_10 in range(N)
:
                                                                for w_snk_2_rank_64 in
range(w_snk_2_rank):
                                                                    summand_7 += (S_90[
src_p_src_6, [i_2_7_214, i_2_2_204, i_2_7_214, i_2_7_214][choose_eps_613_1930], [
i_3_11_222, i_3_8_216, i_3_5_210, i_3_5_210][choose_eps_613_1930], snk_p_snk_10, [[
w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[
w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930], [w_snk_1_spin_0_78[w_snk_2_rank_64
, snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[
w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48]][
choose_eps_613_1930]][choose_i_16_240], [[w_snk_2_color_2_88[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_0_84[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48]][
choose_eps_613_1930], [w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48],
w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64,
snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930]][
choose_i_16_240]] * S_90[src_p_src_6, [i_2_12_224, i_2_12_224, i_2_12_224, i_2_2_204][
choose_eps_613_1930], [i_3_8_216, i_3_11_222, i_3_11_222, i_3_8_216][choose_eps_613_1930
], snk_p_snk_10, [[w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48],
w_snk_2_spin_0_84[w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930], [
```

```
w_snk_1_spin_2_82[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_2_82[w_snk_2_rank_64,
snkSigma2_48], w_snk_1_spin_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_spin_0_78[
w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930]][choose_i_16_240], [[
w_snk_2_color_0_84[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_0_84[
w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930], [w_snk_1_color_2_82[
w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_2_82[w_snk_2_rank_64, snkSigma2_48],
w_snk_1_color_0_78[w_snk_2_rank_64, snkSigma2_48], w_snk_1_color_0_78[w_snk_2_rank_64,
snkSigma2_48]][choose_eps_613_1930]][choose_i_16_240]] * S_90[src_p_src_6, [i_2_2_204,
i_2_7_214, i_2_2_204, i_2_12_224][choose_eps_613_1930], [i_3_5_210, i_3_5_210, i_3_8_216
, i_3_11_222][choose_eps_613_1930], snk_p_snk_10, [[w_snk_2_spin_1_86[w_snk_2_rank_64,
snkSigma2_48], w_snk_2_spin_2_88[w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_2_88[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_spin_1_86[w_snk_2_rank_64, snkSigma2_48]][
choose_eps_613_1930], w_snk_1_spin_1_80[w_snk_2_rank_64, snkSigma2_48]][choose_i_16_240
], [[w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[
w_snk_2_rank_64, snkSigma2_48], w_snk_2_color_2_88[w_snk_2_rank_64, snkSigma2_48],
w_snk_2_color_1_86[w_snk_2_rank_64, snkSigma2_48]][choose_eps_613_1930],
w_snk_1_color_1_80[w_snk_2_rank_64, snkSigma2_48]][choose_i_16_240]] * [w_snk_2_108[
w_snk_2_rank_64, snkSigma2_48], w_snk_1_106[w_snk_2_rank_64, snkSigma2_48]][
choose_i_16_240] * np.conj([phi2_104[snkSpaceRank_32, snk_p_snk_10, snkExternal_16],
phi1_102[snkSpaceRank_32, snk_p_snk_10, snkExternal_16]][choose_i_16_240]))
                                                        merged_vars_baryon_1_3_1948[
choose_eps_613_1930, choose_i_16_240, i_2_12_224, i_2_2_204, i_2_7_214, i_3_11_222,
i_3_5_210, i_3_8_216, snkSigma2_48, snkSpaceRank_32, srcSpaceRank_30, src_p_src_6] = ([
psi2_94[srcSpaceRank_30, src_p_src_6, srcExternal_14], psi1_92[srcSpaceRank_30,
src_p_src_6, srcExternal_14]][choose_i_16_240] * summand_7)
            summand_8 = 0.0
            for srcSpaceRank_30 in range(srcSpaceRank):
                for snkSpaceRank_32 in range(snkSpaceRank):
                    for i_condensed_615_1946 in range(4):
                        for srcSigma1_42 in range(srcSigma1):
                            for srcSigma2_44 in range(srcSigma2):
                                for snkSigma1_46 in range(snkSigma1):
                                    for snkSigma2_48 in range(snkSigma2):
                                        baryon_outer_148_1954: NDArray = np.zeros(1,)
                                        summand_9 = 0.0
                                        for w_src_1_rank_58 in range(w_src_1_rank):
                                            for src_p_src_1952 in range(N):
                                                summand_9 += (w_src_1_96[w_src_1_rank_58
, srcSigma1_42] * merged_vars_baryon_1_3_1948[i_condensed_615_1946, 1, [
w_src_1_spin_2_70[w_src_1_rank_58, srcSigma1_42], w_src_1_spin_2_70[w_src_1_rank_58,
```

```
    srcSigma1_42], w_src_1_spin_2_70[w_src_1_rank_58, srcSigma1_42], w_src_1_spin_1_68[
    w_src_1_rank_58, srcSigma1_42]][i_condensed_615_1946], [w_src_1_spin_1_68[
    w_src_1_rank_58, srcSigma1_42], w_src_1_spin_0_66[w_src_1_rank_58, srcSigma1_42],
    w_src_1_spin_1_68[w_src_1_rank_58, srcSigma1_42], w_src_1_spin_2_70[w_src_1_rank_58,
    srcSigma1_42]][i_condensed_615_1946], [w_src_1_spin_0_66[w_src_1_rank_58, srcSigma1_42],
     w_src_1_spin_1_68[w_src_1_rank_58, srcSigma1_42], w_src_1_spin_0_66[w_src_1_rank_58,
    srcSigma1_42], w_src_1_spin_0_66[w_src_1_rank_58, srcSigma1_42]][i_condensed_615_1946],
    [w_src_1_color_0_66[w_src_1_rank_58, srcSigma1_42], w_src_1_color_2_70[w_src_1_rank_58,
    srcSigma1_42], w_src_1_color_2_70[w_src_1_rank_58, srcSigma1_42], w_src_1_color_1_68[
    w_src_1_rank_58, srcSigma1_42]][i_condensed_615_1946], [w_src_1_color_1_68[
    w_src_1_rank_58, srcSigma1_42], w_src_1_color_1_68[w_src_1_rank_58, srcSigma1_42],
    w_src_1_color_0_66[w_src_1_rank_58, srcSigma1_42], w_src_1_color_0_66[w_src_1_rank_58,
    srcSigma1_42]][i_condensed_615_1946], [w_src_1_color_2_70[w_src_1_rank_58, srcSigma1_42
    ], w_src_1_color_0_66[w_src_1_rank_58, srcSigma1_42], w_src_1_color_1_68[w_src_1_rank_58
    , srcSigma1_42], w_src_1_color_2_70[w_src_1_rank_58, srcSigma1_42]][i_condensed_615_1946
    ], snkExternal_16, snkSigma1_46, snkSpaceRank_32, srcExternal_14, srcSpaceRank_30,
    src_p_src_1952])
                                        baryon_outer_148_1954[None] = summand_9
                                        baryon_outer_149_1956: NDArray = np.zeros(1,)
                                        summand_10 = 0.0
                                        for w_src_2_rank_60 in range(w_src_2_rank):
                                            for src_p_src_1950 in range(N):
                                                summand_10 += (w_src_2_98[
    w_src_2_rank_60, srcSigma2_44] * merged_vars_baryon_1_3_1948[i_condensed_615_1946, 0, [
    w_src_2_spin_0_72[w_src_2_rank_60, srcSigma2_44], w_src_2_spin_1_74[w_src_2_rank_60,
    srcSigma2_44], w_src_2_spin_1_74[w_src_2_rank_60, srcSigma2_44], w_src_2_spin_1_74[
    w_src_2_rank_60, srcSigma2_44]][i_condensed_615_1946], [w_src_2_spin_1_74[
    w_src_2_rank_60, srcSigma2_44], w_src_2_spin_2_76[w_src_2_rank_60, srcSigma2_44],
    w_src_2_spin_2_76[w_src_2_rank_60, srcSigma2_44], w_src_2_spin_2_76[w_src_2_rank_60,
    srcSigma2_44]][i_condensed_615_1946], [w_src_2_spin_2_76[w_src_2_rank_60, srcSigma2_44],
     w_src_2_spin_0_72[w_src_2_rank_60, srcSigma2_44], w_src_2_spin_0_72[w_src_2_rank_60,
    srcSigma2_44], w_src_2_spin_0_72[w_src_2_rank_60, srcSigma2_44]][i_condensed_615_1946],
    [w_src_2_color_2_76[w_src_2_rank_60, srcSigma2_44], w_src_2_color_1_74[w_src_2_rank_60,
    srcSigma2_44], w_src_2_color_1_74[w_src_2_rank_60, srcSigma2_44], w_src_2_color_1_74[
    w_src_2_rank_60, srcSigma2_44]][i_condensed_615_1946], [w_src_2_color_1_74[
    w_src_2_rank_60, srcSigma2_44], w_src_2_color_0_72[w_src_2_rank_60, srcSigma2_44],
    w_src_2_color_0_72[w_src_2_rank_60, srcSigma2_44], w_src_2_color_0_72[w_src_2_rank_60,
    srcSigma2_44]][i_condensed_615_1946], [w_src_2_color_0_72[w_src_2_rank_60, srcSigma2_44
    ], w_src_2_color_2_76[w_src_2_rank_60, srcSigma2_44], w_src_2_color_2_76[w_src_2_rank_60
    , srcSigma2_44], w_src_2_color_2_76[w_src_2_rank_60, srcSigma2_44]][i_condensed_615_1946
    ], snkExternal_16, snkSigma2_48, snkSpaceRank_32, srcExternal_14, srcSpaceRank_30,
```

```
    src_p_src_1950])
                                            baryon_outer_149_1956[None] = summand_10
                                            summand_8 += (v_src_100[srcSigma1_42,
    srcSigma2_44, rhoSrc_22] * v_snk_110[snkSigma1_46, snkSigma2_48, rhoSnk_24] * ([1.0,
    1.0, -1.0, -1.0][i_condensed_615_1946] * [1.0, -1.0, 1.0, -1.0][i_condensed_615_1946] *
    baryon_outer_148_1954[i_condensed_615_1946, snkExternal_16, snkSigma1_46,
    snkSpaceRank_32, srcExternal_14, srcSigma1_42, srcSpaceRank_30] * baryon_outer_149_1956[
    i_condensed_615_1946, snkExternal_16, snkSigma2_48, snkSpaceRank_32, srcExternal_14,
    srcSigma2_44, srcSpaceRank_30]))
                sep_unroll_perm_0_7330295488186759212_baryon_145_sumed_1960[None] =
    summand_8
                output_2080[snkExternal_16, rhoSnk_24, srcExternal_14, rhoSrc_22] = (
    sep_unroll_perm_2_3510658113350937887_epsilon_144_sumed_1958[rhoSnk_24, snkExternal_16,
    srcExternal_14, rhoSrc_22] + sep_unroll_perm_0_7330295488186759212_baryon_145_sumed_1960
    [rhoSnk_24, snkExternal_16, srcExternal_14, rhoSrc_22])
```

Listing A.3: Dibaryon Dibaryon Rewritten IR (prcomputes removed for brevity)

A.3 Dibaryon Hexaquark IR

```
output_92: NDArray = np.zeros(rhoSrcSize, EN, EN, rhoSnkHSize)
for rhoSrc_20 in range(rhoSrcSize):
    for srcExternal_12 in range(EN):
        for snkExternal_14 in range(EN):
            for rhoHSnk_22 in range(rhoSnkHSize):
                summand_0 = 0.0
                for srcSigma1_32 in range(srcSigma1):
                    for w_src_1_rank_42 in range(w_src_1_rank):
                        for snk_snk_8 in range(N):
                            for src_src_4 in range(N):
                                for srcSigma2_34 in range(srcSigma2):
                                    for src_p_src_6 in range(N):
                                        for w_src_2_rank_44 in range(
                                                w_src_2_rank):
                                            for w_snk_H_rank_46 in range(
                                                    w_snk_H_rank):
                                                for srcSpaceRank_26 in range(
                                                        srcSpaceRank):
                                                    for up_88 in itertools.permutations(
```

```
                                                    range(3)):
                                                for down_90 in itertools.
    permutations(
                                                        range(3)):
                                                    summand_0 += (
                                                        S_72[
                                                            src_src_4,
                                                            w_src_1_spin_0_48[
                                                                w_src_1_rank_42,
                                                                srcSigma1_32],
                                                            w_src_1_color_0_48[
                                                                w_src_1_rank_42,
                                                                srcSigma1_32],
                                                            [
                                                                snk_snk_8,
                                                                snk_snk_8,
                                                                snk_snk_8
                                                            ][up_88[0]],
                                                            [
                                                                w_snk_spin_0_60[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_spin_2_64[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_spin_4_68[
                                                                    w_snk_H_rank_46]
                                                            ][up_88[0]],
                                                            [
                                                                w_snk_color_0_60[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_2_64[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_4_68[
                                                                    w_snk_H_rank_46]
                                                            ][up_88[0]]]
                                                        *
                                                        S_72[src_src_4,
                                                             w_src_1_spin_2_52[
```

```
                                                                    w_src_1_rank_42,
                                                                    srcSigma1_32],
                                                                w_src_1_color_2_52[
                                                                    w_src_1_rank_42,
                                                                    srcSigma1_32],
                                                                [
                                                                    snk_snk_8,
                                                                    snk_snk_8,
                                                                    snk_snk_8
                                                                ][up_88[
                                                                    1]],
                                                                [
                                                                    w_snk_spin_0_60[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_spin_2_64[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_spin_4_68[
                                                                        w_snk_H_rank_46
    ]
                                                                ][up_88[
                                                                    1]],
                                                                [
                                                                    w_snk_color_0_60[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_color_2_64[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_color_4_68[
                                                                        w_snk_H_rank_46
    ]
                                                                ][up_88[
                                                                    1]]] *
                                                            S_72[src_p_src_6,
                                                                w_src_2_spin_1_56[
                                                                    w_src_2_rank_44,
                                                                    srcSigma2_34],
                                                                w_src_2_color_1_56[
                                                                    w_src_2_rank_44,
```

```
                                                                srcSigma2_34],
                                                            [
                                                                snk_snk_8,
                                                                snk_snk_8,
                                                                snk_snk_8
                                                            ][up_88[
                                                                2]],
                                                            [
                                                                w_snk_spin_0_60[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_spin_2_64[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_spin_4_68[
                                                                    w_snk_H_rank_46
    ]
                                                            ][up_88[
                                                                2]],
                                                            [
                                                                w_snk_color_0_60[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_2_64[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_4_68[
                                                                    w_snk_H_rank_46
    ]
                                                            ][up_88[
                                                                2]]] *
                                                    sign(up_88)
                                                    * S_72[
                                                        src_src_4,
                                                        w_src_1_spin_1_50[
                                                            w_src_1_rank_42,
                                                            srcSigma1_32],
                                                        w_src_1_color_1_50[
                                                            w_src_1_rank_42,
                                                            srcSigma1_32],
                                                        [
```

```
                                                                    snk_snk_8,
                                                                    snk_snk_8,
                                                                    snk_snk_8
                                                                ][down_90[
                                                                    0]],
                                                                [
                                                                    w_snk_spin_1_62[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_spin_3_66[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_spin_5_70[
                                                                        w_snk_H_rank_46]
                                                                ][down_90[
                                                                    0]],
                                                                [
                                                                    w_snk_color_1_62[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_color_3_66[
                                                                        w_snk_H_rank_46
    ],
                                                                    w_snk_color_5_70[
                                                                        w_snk_H_rank_46]
                                                                ][down_90[
                                                                    0]]] *
                                                            S_72[
                                                                src_p_src_6,
                                                                w_src_2_spin_0_54[
                                                                    w_src_2_rank_44,
                                                                    srcSigma2_34],
                                                                w_src_2_color_0_54[
                                                                    w_src_2_rank_44,
                                                                    srcSigma2_34],
                                                                [
                                                                    snk_snk_8,
                                                                    snk_snk_8,
                                                                    snk_snk_8
                                                                ][down_90[
                                                                    1]],
```

```
                                                    [
                                                        w_snk_spin_1_62[
                                                            w_snk_H_rank_46
    ],
                                                        w_snk_spin_3_66[
                                                            w_snk_H_rank_46
    ],
                                                        w_snk_spin_5_70[
                                                            w_snk_H_rank_46]
                                                    ][down_90[
                                                        1]],
                                                    [
                                                        w_snk_color_1_62[
                                                            w_snk_H_rank_46
    ],
                                                        w_snk_color_3_66[
                                                            w_snk_H_rank_46
    ],
                                                        w_snk_color_5_70[
                                                            w_snk_H_rank_46]
                                                    ][down_90[
                                                        1]]] *
                                                S_72[
                                                    src_p_src_6,
                                                    w_src_2_spin_2_58[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34],
                                                    w_src_2_color_2_58[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34],
                                                    [
                                                        snk_snk_8,
                                                        snk_snk_8,
                                                        snk_snk_8
                                                    ][down_90[
                                                        2]],
                                                    [
                                                        w_snk_spin_1_62[
                                                            w_snk_H_rank_46
    ],
                                                        w_snk_spin_3_66[
```

```
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_spin_5_70[
                                                                    w_snk_H_rank_46]
                                                            ][down_90[
                                                                2]],
                                                            [
                                                                w_snk_color_1_62[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_3_66[
                                                                    w_snk_H_rank_46
    ],
                                                                w_snk_color_5_70[
                                                                    w_snk_H_rank_46]
                                                            ][down_90[
                                                                2]]] *
                                                        sign(down_90) *
                                                        (psi1_74[
                                                            srcSpaceRank_26,
                                                            src_src_4,
                                                            srcExternal_12]
                                                         * psi2_76[
                                                             srcSpaceRank_26,
                                                             src_p_src_6,
                                                             srcExternal_12]
                                                         * w_src_1_78[
                                                             w_src_1_rank_42,
                                                             srcSigma1_32]
                                                         * w_src_2_80[
                                                             w_src_2_rank_44,
                                                             srcSigma2_34]
                                                         * v_src_82[
                                                             srcSigma1_32,
                                                             srcSigma2_34,
                                                             rhoSrc_20]
                                                         ) * np.
                                                        conj((phi_84[
                                                            snk_snk_8,
                                                            snkExternal_14]
                                                              *
```

```
                                                                    w_snk_86[
                                                                        w_snk_H_rank_46,
                                                                        rhoHSnk_22]
                                                                    )))
                output_92[rhoSrc_20, srcExternal_12, snkExternal_14,
                          rhoHSnk_22] = summand_0
```

Listing A.4: Dibaryon Hexaquark Naive IR

```
output_104: NDArray = np.zeros(EN, EN, rhoSrcSize, rhoSnkHSize)
for srcExternal_12 in range(EN):
    for snkExternal_14 in range(EN):
        for rhoSrc_20 in range(rhoSrcSize):
            for rhoHSnk_22 in range(rhoSnkHSize):
                summand_0 = 0.0
                for srcSpaceRank_26 in range(srcSpaceRank):
                    for snk_snk_8 in range(N):
                        for srcSigma1_32 in range(srcSigma1):
                            for w_snk_H_rank_46 in range(w_snk_H_rank):
                                for srcSigma2_34 in range(srcSigma2):
                                    for up_88 in itertools.permutations(
                                            range(3)):
                                        for down_90 in itertools.permutations(
                                                range(3)):
                                            part_0_2_102: NDArray = np.zeros(2)
                                            for choose_i_0_98 in range(2):
                                                summand_1 = 0.0
                                                for src_p_src_6 in range(N):
                                                    for w_src_2_rank_44 in range(
                                                            w_src_2_rank):
                                                        summand_1 += (S_72[src_p_src_6, [
                                                            w_src_2_spin_1_56[
                                                                w_src_2_rank_44,
                                                                srcSigma2_34],
                                                            w_src_1_spin_0_48[
                                                                w_src_2_rank_44,
                                                                srcSigma2_34]
                                                        ][choose_i_0_98], [
                                                            w_src_2_color_1_56[
                                                                w_src_2_rank_44,
                                                                srcSigma2_34],
                                                            w_src_1_color_0_48[
```

```
                                                        w_src_2_rank_44,
                                                        srcSigma2_34]
                                                    ][choose_i_0_98], snk_snk_8, [
                                                        w_snk_spin_0_60[
                                                            w_snk_H_rank_46],
                                                        w_snk_spin_2_64[
                                                            w_snk_H_rank_46],
                                                        w_snk_spin_4_68[
                                                            w_snk_H_rank_46]
                                                    ][up_88[[
                                                        2, 0
                                                    ][choose_i_0_98]]], [
                                                        w_snk_color_0_60[
                                                            w_snk_H_rank_46],
                                                        w_snk_color_2_64[
                                                            w_snk_H_rank_46],
                                                        w_snk_color_4_68[
                                                            w_snk_H_rank_46]
                                                    ][up_88[[
                                                        2, 0
                                                    ][choose_i_0_98]]]] * S_72[
    src_p_src_6, [
                                                        w_src_2_spin_0_54[
                                                            w_src_2_rank_44,
                                                            srcSigma2_34],
                                                        w_src_1_spin_2_52[
                                                            w_src_2_rank_44,
                                                            srcSigma2_34]
                                                    ][choose_i_0_98], [
                                                        w_src_2_color_0_54[
                                                            w_src_2_rank_44,
                                                            srcSigma2_34],
                                                        w_src_1_color_2_52[
                                                            w_src_2_rank_44,
                                                            srcSigma2_34]
                                                    ][choose_i_0_98], snk_snk_8, [
                                                        [
                                                            w_snk_spin_1_62[
                                                                w_snk_H_rank_46],
                                                            w_snk_spin_0_60[
                                                                w_snk_H_rank_46]
```

```
                                                    ][choose_i_0_98],
                                                    [
                                                        w_snk_spin_3_66[
                                                            w_snk_H_rank_46],
                                                        w_snk_spin_2_64[
                                                            w_snk_H_rank_46]
                                                    ][choose_i_0_98],
                                                    [
                                                        w_snk_spin_5_70[
                                                            w_snk_H_rank_46],
                                                        w_snk_spin_4_68[
                                                            w_snk_H_rank_46]
                                                    ][choose_i_0_98]
                                                ][[
                                                    down_90[1], up_88[1]
                                                ][choose_i_0_98]], [
                                                    [
                                                        w_snk_color_1_62[
                                                            w_snk_H_rank_46],
                                                        w_snk_color_0_60[
                                                            w_snk_H_rank_46]
                                                    ][choose_i_0_98],
                                                    [
                                                        w_snk_color_3_66[
                                                            w_snk_H_rank_46],
                                                        w_snk_color_2_64[
                                                            w_snk_H_rank_46]
                                                    ][choose_i_0_98],
                                                    [
                                                        w_snk_color_5_70[
                                                            w_snk_H_rank_46],
                                                        w_snk_color_4_68[
                                                            w_snk_H_rank_46]
                                                    ][choose_i_0_98]
                                                ][[
                                                    down_90[1], up_88[1]
                                                ][choose_i_0_98]]] * S_72[
    src_p_src_6, [
                                                    w_src_2_spin_2_58[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34],
```

```
                                                    w_src_1_spin_1_50[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34]
                                                ][choose_i_0_98], [
                                                    w_src_2_color_2_58[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34],
                                                    w_src_1_color_1_50[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34]
                                                ][choose_i_0_98], snk_snk_8, [
                                                    w_snk_spin_1_62[
                                                        w_snk_H_rank_46],
                                                    w_snk_spin_3_66[
                                                        w_snk_H_rank_46],
                                                    w_snk_spin_5_70[
                                                        w_snk_H_rank_46]
                                                ][down_90[[
                                                    2, 0
                                                ][choose_i_0_98]]], [
                                                    w_snk_color_1_62[
                                                        w_snk_H_rank_46],
                                                    w_snk_color_3_66[
                                                        w_snk_H_rank_46],
                                                    w_snk_color_5_70[
                                                        w_snk_H_rank_46]
                                                ][down_90[[
                                                    2, 0
                                                ][choose_i_0_98]]]] * [
                                                    psi2_76[
                                                        srcSpaceRank_26,
                                                        src_p_src_6,
                                                        srcExternal_12],
                                                    psi1_74[
                                                        srcSpaceRank_26,
                                                        src_p_src_6,
                                                        srcExternal_12]
                                                ][choose_i_0_98] * [
                                                    w_src_2_80[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34],
```

```
                                                    w_src_1_78[
                                                        w_src_2_rank_44,
                                                        srcSigma2_34]
                                                ][choose_i_0_98])
                                        part_0_2_102[
                                            choose_i_0_98] = summand_1
                                    summand_0 += (
                                        sign(up_88) * sign(down_90) *
                                        v_src_82[srcSigma1_32,
                                                 srcSigma2_34,
                                                 rhoSrc_20] *
                                        np.conj(phi_84[snk_snk_8,
                                                       snkExternal_14])
                                        * np.conj(
                                            w_snk_86[w_snk_H_rank_46,
                                                     rhoHSnk_22]) *
                                        (part_0_2_102[
                                            0, down_90, snk_snk_8,
                                            srcExternal_12,
                                            srcSigma2_34,
                                            srcSpaceRank_26, up_88,
                                            w_snk_H_rank_46] *
                                         part_0_2_102[
                                             1, down_90, snk_snk_8,
                                             srcExternal_12,
                                             srcSigma1_32,
                                             srcSpaceRank_26, up_88,
                                             w_snk_H_rank_46]))
                output_104[srcExternal_12, snkExternal_14, rhoSrc_20,
                           rhoHSnk_22] = summand_0
```

Listing A.5: Dibaryon Hexaquark Rewritten IR

A.4 Hexaquark Hexaquark IR

```
output_68: NDArray = np.zeros(EN, EN, rhoSrcHSize, rhoSrcHSize)
for srcExternal_14 in range(EN):
    for snkExternal_16 in range(EN):
        for rhoHSnk_20 in range(rhoSrcHSize):
```

```
            for rhoHSrc_18 in range(rhoSrcHSize):
                summand_0 = 0.0
                for w_snk_1_rank_52 in range(w_snk_1_rank):
                    for snk_snk_6 in range(N):
                        for w_src_1_rank_50 in range(w_src_1_rank):
                            for src_src_4 in range(N):
                                for up_64 in itertools.permutations(range(3)):
                                    for down_66 in itertools.permutations(
                                            range(3)):
                                        summand_0 += (S_54[
                                            src_src_4,
                                            w_src_spin_0_22[w_src_1_rank_50],
                                            w_src_color_0_22[w_src_1_rank_50],
                                            [snk_snk_6, snk_snk_6, snk_snk_6][
                                                up_64[0]],
                                            [
                                                w_snk_spin_0_34[
                                                    w_snk_1_rank_52],
                                                w_snk_spin_2_38[
                                                    w_snk_1_rank_52],
                                                w_snk_spin_4_42[w_snk_1_rank_52]
                                            ][up_64[0]], [
                                                w_snk_color_0_34[
                                                    w_snk_1_rank_52],
                                                w_snk_color_2_38[
                                                    w_snk_1_rank_52],
                                                w_snk_color_4_42[
                                                    w_snk_1_rank_52]
                                            ][up_64[0]]] * S_54[
                                                src_src_4, w_src_spin_2_26[
                                                    w_src_1_rank_50],
                                                w_src_color_2_26[
                                                    w_src_1_rank_50], [
                                                        snk_snk_6, snk_snk_6,
                                                        snk_snk_6
                                                    ][up_64[1]], [
                                                        w_snk_spin_0_34[
                                                            w_snk_1_rank_52],
                                                        w_snk_spin_2_38[
                                                            w_snk_1_rank_52],
                                                        w_snk_spin_4_42[
```

```
                                                    w_snk_1_rank_52]
                                                ][up_64[1]],
                                            [
                                                w_snk_color_0_34[
                                                    w_snk_1_rank_52],
                                                w_snk_color_2_38[
                                                    w_snk_1_rank_52],
                                                w_snk_color_4_42[
                                                    w_snk_1_rank_52]
                                            ][up_64[1]]] * S_54[
                                                src_src_4, w_src_spin_4_30[
                                                    w_src_1_rank_50],
                                                w_src_color_4_30[
                                                    w_src_1_rank_50],
                                                [
                                                    snk_snk_6, snk_snk_6,
                                                    snk_snk_6
                                                ][up_64[2]], [
                                                    w_snk_spin_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_4_42[
                                                        w_snk_1_rank_52]
                                                ][up_64[2]], [
                                                    w_snk_color_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_4_42[
                                                        w_snk_1_rank_52]
                                                ][up_64[2]]] * sign(up_64) * S_54[
                                                    src_src_4,
                                                    w_src_spin_1_24[
                                                        w_src_1_rank_50],
                                                    w_src_color_1_24[
                                                        w_src_1_rank_50],
                                                    [
                                                        snk_snk_6,
                                                        snk_snk_6, snk_snk_6
                                                    ][down_66[
```

```
                                                    0]],
                                                [
                                                    w_snk_spin_1_36[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_3_40[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_5_44[
                                                        w_snk_1_rank_52]
                                                ][down_66[0]], [
                                                    w_snk_color_1_36[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_3_40[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_5_44[
                                                        w_snk_1_rank_52]
                                                ][down_66[0]]] *
                                            S_54[
                                                src_src_4,
                                                w_src_spin_3_28[
                                                    w_src_1_rank_50],
                                                w_src_color_3_28[
                                                    w_src_1_rank_50],
                                                [
                                                    snk_snk_6,
                                                    snk_snk_6,
                                                    snk_snk_6
                                                ][down_66[1]], [
                                                    w_snk_spin_1_36[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_3_40[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_5_44[
                                                        w_snk_1_rank_52]
                                                ][down_66[1]], [
                                                    w_snk_color_1_36[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_3_40[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_5_44[
                                                        w_snk_1_rank_52]
                                                ][down_66[1]]] *
```

```
                                                S_54[
                                                    src_src_4,
                                                    w_src_spin_5_32[
                                                        w_src_1_rank_50],
                                                    w_src_color_5_32[
                                                        w_src_1_rank_50],
                                                    [
                                                        snk_snk_6,
                                                        snk_snk_6,
                                                        snk_snk_6
                                                    ][down_66[2]], [
                                                        w_snk_spin_1_36[
                                                            w_snk_1_rank_52],
                                                        w_snk_spin_3_40[
                                                            w_snk_1_rank_52],
                                                        w_snk_spin_5_44[
                                                            w_snk_1_rank_52]
                                                    ][down_66[2]], [
                                                        w_snk_color_1_36[
                                                            w_snk_1_rank_52],
                                                        w_snk_color_3_40[
                                                            w_snk_1_rank_52],
                                                        w_snk_color_5_44[
                                                            w_snk_1_rank_52]
                                                    ][down_66[2]]] *
                                                sign(down_66) *
                                                (psi1_56[src_src_4,
                                                         srcExternal_14]
                                                 * w_src_58[
                                                     w_src_1_rank_50,
                                                     rhoHSrc_18]) *
                                                np.conj(
                                                    (phi1_60[
                                                        snk_snk_6,
                                                        snkExternal_16] *
                                                     w_snk_62[
                                                         w_snk_1_rank_52,
                                                         rhoHSnk_20])))
                output_68[srcExternal_14, snkExternal_16, rhoHSnk_20,
```

```
                            rhoHSrc_18] = summand_0
```

Listing A.6: Hexaquark Hexaquark Naive IR

```
output_68: NDArray = np.zeros(EN, rhoSrcHSize, rhoSrcHSize, EN)
for snkExternal_16 in range(EN):
    for rhoHSrc_18 in range(rhoSrcHSize):
        for rhoHSnk_20 in range(rhoSrcHSize):
            for srcExternal_14 in range(EN):
                summand_0 = 0.0
                for w_snk_1_rank_52 in range(w_snk_1_rank):
                    for w_src_1_rank_50 in range(w_src_1_rank):
                        for snk_snk_6 in range(N):
                            for src_src_4 in range(N):
                                for up_64 in itertools.permutations(range(3)):
                                    for down_66 in itertools.permutations(
                                            range(3)):
                                        summand_0 += (
                                            S_54[src_src_4, w_src_spin_0_22[
                                                w_src_1_rank_50],
                                                 w_src_color_0_22[
                                                     w_src_1_rank_50],
                                                 snk_snk_6, [
                                                     w_snk_spin_0_34[
                                                         w_snk_1_rank_52],
                                                     w_snk_spin_2_38[
                                                         w_snk_1_rank_52],
                                                     w_snk_spin_4_42[
                                                         w_snk_1_rank_52]
                                                 ][up_64[0]], [
                                                     w_snk_color_0_34[
                                                         w_snk_1_rank_52],
                                                     w_snk_color_2_38[
                                                         w_snk_1_rank_52],
                                                     w_snk_color_4_42[
                                                         w_snk_1_rank_52]
                                                 ][up_64[0]]] *
                                            S_54[src_src_4, w_src_spin_2_26[
                                                w_src_1_rank_50],
                                                 w_src_color_2_26[
                                                     w_src_1_rank_50],
                                                 snk_snk_6, [
```

```
                                                    w_snk_spin_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_4_42[
                                                        w_snk_1_rank_52]
                                                ][up_64[1]], [
                                                    w_snk_color_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_4_42[
                                                        w_snk_1_rank_52]
                                                ][up_64[1]]] *
                                            S_54[src_src_4, w_src_spin_4_30[
                                                w_src_1_rank_50],
                                                 w_src_color_4_30[
                                                    w_src_1_rank_50],
                                                 snk_snk_6, [
                                                    w_snk_spin_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_spin_4_42[
                                                        w_snk_1_rank_52]
                                                 ][up_64[2]], [
                                                    w_snk_color_0_34[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_2_38[
                                                        w_snk_1_rank_52],
                                                    w_snk_color_4_42[
                                                        w_snk_1_rank_52]
                                                 ][up_64[2]]] * sign(up_64)
                                            * S_54[src_src_4, w_src_spin_1_24[
                                                w_src_1_rank_50],
                                                   w_src_color_1_24[
                                                       w_src_1_rank_50],
                                                   snk_snk_6, [
                                                       w_snk_spin_1_36[
                                                           w_snk_1_rank_52],
                                                       w_snk_spin_3_40[
```

```
                                                    w_snk_1_rank_52],
                                                w_snk_spin_5_44[
                                                    w_snk_1_rank_52]
                                           ][down_66[0]], [
                                                w_snk_color_1_36[
                                                    w_snk_1_rank_52],
                                                w_snk_color_3_40[
                                                    w_snk_1_rank_52],
                                                w_snk_color_5_44[
                                                    w_snk_1_rank_52]
                                           ][down_66[0]]] *
                                    S_54[src_src_4, w_src_spin_3_28[
                                        w_src_1_rank_50],
                                         w_src_color_3_28[
                                             w_src_1_rank_50],
                                         snk_snk_6, [
                                             w_snk_spin_1_36[
                                                 w_snk_1_rank_52],
                                             w_snk_spin_3_40[
                                                 w_snk_1_rank_52],
                                             w_snk_spin_5_44[
                                                 w_snk_1_rank_52]
                                         ][down_66[1]], [
                                             w_snk_color_1_36[
                                                 w_snk_1_rank_52],
                                             w_snk_color_3_40[
                                                 w_snk_1_rank_52],
                                             w_snk_color_5_44[
                                                 w_snk_1_rank_52]
                                         ][down_66[1]]] *
                                    S_54[src_src_4, w_src_spin_5_32[
                                        w_src_1_rank_50],
                                         w_src_color_5_32[
                                             w_src_1_rank_50],
                                         snk_snk_6, [
                                             w_snk_spin_1_36[
                                                 w_snk_1_rank_52],
                                             w_snk_spin_3_40[
                                                 w_snk_1_rank_52],
                                             w_snk_spin_5_44[
                                                 w_snk_1_rank_52]
```

```
                                            ][down_66[2]], [
                                                w_snk_color_1_36[
                                                    w_snk_1_rank_52],
                                                w_snk_color_3_40[
                                                    w_snk_1_rank_52],
                                                w_snk_color_5_44[
                                                    w_snk_1_rank_52]
                                            ][down_66[2]]] *
                                        sign(down_66) *
                                        (psi1_56[src_src_4, srcExternal_14]
                                         * w_src_58[w_src_1_rank_50,
                                                    rhoHSrc_18]) *
                                        np.conj((phi1_60[snk_snk_6,
                                                         snkExternal_16] *
                                                 w_snk_62[w_snk_1_rank_52,
                                                          rhoHSnk_20])))
            output_68[snkExternal_16, rhoHSrc_18, rhoHSnk_20,
                      srcExternal_14] = summand_0
```

Listing A.7: Hexaquark Hexaquark Rewritten IR

Appendix B

Rewrite Code

B.1 Dibaryon Dibaron Rewrites

```
def dibar_rewrite(comp):
    up_perm = find_index_by_name(comp, 'up')
    down_perm = find_index_by_name(comp, 'down')

    res = simplify_conj(comp, ['w_src_1', 'w_src_2', 'w_snk_1', 'w_snk_2',
    'v_src', 'v_snk'])
    res = separate_sum(res, ["srcSpaceRank", "snkSpaceRank"], move_vars=
   True)

    # Expand the perms
    def expand_perms(exp):
        exp = expand_perm(exp, up_perm)
        exp = apply_to_all_lets(exp, lambda x: propagate_const_acc(
   expand_perm(x, down_perm)))
        exp = flatten_multilets(exp)
        return exp

    res = run_on_loc(res, [2], expand_perms)
```

```
16
17      def reduce_space_loops(perm_let):
18          perm_letp = run_on_loc(perm_let, [], lambda x: separate_sum(x, ['
    w_src_1_rank', 'w_src_2_rank', 'srcSigma1', 'srcSigma2', 'up', 'down'],
     move_vars=True))
19
20          groups = ['w_src_1_rank', 'w_src_2_rank', 'srcSigma1', 'srcSigma2'
    ]
21          perm_letpp = run_on_loc(perm_letp, [], lambda y:
    loop_linearize_controllable(y, groups, {'w_src_1_rank': 12, '
    w_src_2_rank': 12, 'srcSigma1': 2, 'srcSigma2': 2}))
22
23          perm_letppp = run_on_loc(perm_letpp, [0, 2, 0, 1], lambda y:
    separate_sum(y, ["snkSigma1", "snkSigma2"], move_vars=True))
24          key = [0, 2, 0, 1, 2, 1]
25          if not can_partition(get_from_ir(perm_letppp, key)):
26              perm_letppp = run_on_loc(perm_letppp, key, lambda x:
    separate_sum(x, ['src_src', 'src_p_src'], move_vars=True))
27              key = [0, 2, 0, 1, 2, 1, 2, 2, 0, 0]
28              assert can_partition(get_from_ir(perm_letppp, key))
29              perm_letppp_parted1 = run_on_loc(perm_letppp, key, lambda x:
    partition_exprs(x, multi=True, name="epsilon"))
30              perm_letppp_merged = run_on_loc(perm_letppp_parted1, key,
    merge_multilet)
31              return perm_letppp_merged
32          else:
33              temp = run_on_loc(perm_letppp, key, lambda x: partition_exprs(
    x, name="baryon"))
34              tempp = run_on_loc(temp, key, merge_multilet)
35              temppp = run_on_loc(tempp, key + [0,], lambda y: separate_sum(
    y, ["src_p_src", "src_src"], move_vars=True))
36              return temppp
37
```

```
38     res = run_on_loc(res, [2], lambda x: apply_to_all_lets(x,
       reduce_space_loops))
39
40     # Group by number of spatial loops.
41     res = run_on_loc(res, [2], lambda x: separate_lets(x, lambda y:
       count_num_sums(y, ['src_src', 'src_p_src', 'snk_snk', 'snk_p_snk']) ==
       2, {True : "baryon", False : "epsilon"}))
42
43     res = run_on_loc(res, [2], lambda x: apply_to_all_lets(x, lambda y:
       merge_multilet(y, name="eps")))
44     res = run_on_loc(res, [2], lambda x: apply_to_all_lets(x, lambda y:
       run_on_loc(y, [2], condense_add)))
45         pass
46
47     res = run_on_loc(res, [2, 0, 0, 1], push_use_into_let)
48     res = run_on_loc(res, [2, 0, 0, 1, 2], push_use_into_let)
49     res = run_on_loc(res, [2, 0, 0, 1], merge_sum)
50     res = run_on_loc(res, [2, 0, 0, 1], merge_sum)
51     res = run_on_loc(res, [2, 0, 0, 1], merge_sum)
52
53     res = run_on_loc(res, [2, 0, 1, 1], push_use_into_let)
54     res = run_on_loc(res, [2, 0, 1, 1, 2], push_use_into_let)
55     res = run_on_loc(res, [2, 0, 1, 1], merge_sum)
56     res = run_on_loc(res, [2, 0, 1, 1], merge_sum)
57
58     res = run_on_loc(res, [2, 0, 1, 1, 2, 2, 0, 2], pull_sum_from_let)
59
60     res = run_on_loc(res, [2, 0, 1, 1], move_into_let_use)
61     res = (run_on_loc(res, [2, 0, 1, 1, 2], lambda y: separate_sum(y, ["
       w_src_1_rank", "w_src_2_rank"], move_vars=True, tonotraise=True)))
62
63     assert can_partition(get_from_ir(res, [2, 0, 1, 1, 2, 2, 2, 0, 0]))
64     res = run_on_loc(res, [2, 0, 1, 1, 2, 2, 2, 0, 0], lambda x:
```

```
partition_exprs(x, multi=False, homo=False, name="baryon_outer"))
 res = run_on_loc(res, [2, 0, 1, 1, 2, 2, 2, 0, 0, 2, 0, 0, 0],
merge_sum)
 res = run_on_loc(res, [2, 0, 1, 1, 2, 2, 2, 0, 0, 2, 0, 1, 0],
merge_sum)

 res = push_sum_varaccs(res)
 res = run_on_loc(res, [1,], push_sum_to_acceses)
 res = push_sum_varaccs(res)
 res = run_on_loc(res, [0, 0, 1], merge_sum)
 res = run_on_loc(res, [0, 1, 1], move_into_let_use)
 res = run_on_loc(res, [0, 1, 1], move_into_let_use)
 res = run_on_loc(res, [0, 1, 1, 2], merge_sum)
 res = (run_on_loc(res, [0, 0, 1], lambda y: separate_sum(y, ['
src_p_src',  'src_src', 'srcSpaceRank'], move_vars=True, tonotraise=
False)))
 res = (run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2],
normalize_let_uses))
 res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0], lambda y:
separate_sum(y, ['snk_p_snk'], move_vars=True))

 # Pushes the choice on the linearized indicies to another level
 res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
transfer_index_logic_to_use(y, ["i_2", "i_3"], name="sc"))
 # Resimplfies access to that
 res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2],
resimplify_choice)

 res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2],   lambda x:
unroll_let_expr(x, "choose_i", name="merged_eps", multi=True))
 # Condense the choice
 res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
condenseChoiceVerticalFilter(y, "merged_eps_0", "choose_ep", filter =
```

```
    lambda x: len(set(x.vars)) == 2 and not isinstance(x.vars[0].access,
    list | tuple) and x.vars[0].access.iname ==  "src_p_src"))
87  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    condenseChoiceVerticalFilter(y, "merged_eps_1", "choose_ep", filter =
    lambda x: len(set(x.vars)) == 2 and not isinstance(x.vars[0].access,
    list | tuple) and x.vars[0].access.iname ==  "src_p_src"))
88  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    condenseChoices(y, "merged_eps_0_compressed", "choose_eps", idxMatcher=
    lambda x: "compressed" not in x))
89  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    condenseChoices(y, "merged_eps_1_compressed", "choose_eps", idxMatcher=
    lambda x: "compressed" not in x))
90  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0, 1, 1],
    clear_dead_multi)
91  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0, 1, 1],
    lambda y: transfer_index_logic_to_use_new(y, ["choose_eps"]))
92  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0, 0, 1],
    clear_dead_multi)
93  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 0, 0, 1],
    lambda y: transfer_index_logic_to_use_new(y, ["choose_eps"]))
94  res = run_on_loc(res, [0, 0, 1,2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    transfer_index_logic_to_use_new(y, ["choose_eps"], idxLet=0))
95  res = run_on_loc(res, [0, 0, 1,2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    transfer_index_logic_to_use_new(y, ["choose_eps"], idxLet=1))
96  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2], lambda y:
    precompute_access(y, ["w_src_2_rank", "i_condensed_614"]))
97
98  res = run_on_loc(res, [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 1, 0, 0, 1, 0,
     0, 1, 0, 0, 1, 2, 1, 2], lambda y:  precompute_access(y, ["
    w_snk_2_rank"]))
99  res = run_on_loc(res,  [0, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 1, 0, 1, 1,
    0, 0, 1, 0, 0, 1, 2, 1, 2], lambda y:  precompute_access(y, ["
    w_snk_2_rank"]))
```

```
100
101     # Partition that last sum - do the reduction seperately
102     res = apply_to_all_til_no_change(res, LQCD_IR.MultiLet,
    clear_dead_multi)
103     res = apply_to_all_til_no_change(res, LQCD_IR.Let | LQCD_IR.MultiLet |
     LQCD_IR.Sum, clean_unneeded_binds)
104     return res
```

Listing B.1: Dibaryon Dibaryon Rewrites

References

[1] R. Baghdadi, J. Ray, M. B. Romdhane, E. D. Sozzo, A. Akkas, Y. Zhang, P. Suriana, S. Kamil, and S. P. Amarasinghe, "Tiramisu: A polyhedral compiler for expressing fast and portable code," *CoRR*, vol. abs/1804.10694, 2018. arXiv: 1804.10694. [Online]. Available: http://arxiv.org/abs/1804.10694.

[2] J. Ragan-Kelley, A. Adams, D. Sharlet, C. Barnes, S. Paris, M. Levoy, S. Amarasinghe, and F. Durand, "Halide: Decoupling algorithms from schedules for high-performance image processing," *Commun. ACM*, vol. 61, no. 1, pp. 106–115, Dec. 2017, ISSN: 0001-0782. DOI: 10.1145/3150211. [Online]. Available: https://doi.org/10.1145/3150211.

[3] C. Gattringer and C. B. Lang, *Quantum Chromodynamics on the Lattice: An Introductory Presentation* (Lecture Notes in Physics). Springer, 2010, vol. 788. DOI: 10.1007/978-3-642-01850-3.

[4] K. G. Wilson, "Confinement of quarks," *Phys. Rev. D*, vol. 10, pp. 2445–2459, 8 Oct. 1974. DOI: 10.1103/PhysRevD.10.2445. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.10.2445.

[5] R. Gupta, *Introduction to lattice qcd*, 1998. arXiv: hep-lat/9807028 [hep-lat].

[6] W. Detmold and K. Orginos, "Nuclear correlation functions in lattice qcd," *Phys. Rev. D*, vol. 87, p. 114512, 11 Jun. 2013. DOI: 10.1103/PhysRevD.87.114512. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.87.114512.

[7] S. Amarasinghe, R. Baghdadi, Z. Davoudi, W. Detmold, M. Illa, A. Parreño, A. V. Pochinsky, P. E. Shanahan, and M. L. Wagman, "Variational study of two-nucleon systems with lattice qcd," *Physical Review D*, vol. 107, no. 9, May 2023, ISSN: 2470-0029. DOI: 10.1103/physrevd.107.094508. [Online]. Available: http://dx.doi.org/10.1103/PhysRevD.107.094508.

[8] G. Baumgartner, A. Auer, D. Bernholdt, *et al.*, "Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 276–292, 2005. DOI: 10.1109/JPROC.2004.840311.

[9] J. A. Mason, *Learning APL: An Array Processing Language*. USA: John Wiley & Sons, Inc., 1985, ISBN: 0471603392.

[10] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, ser. SIGFIDET '74, Ann Arbor, Michigan: Association for Computing Machinery, 1974, pp. 249–264, ISBN: 9781450374156. DOI: 10.1145/800296.811515. [Online]. Available: https://doi.org/10.1145/800296.811515.

[11] R. T. Mullapudi, A. Adams, D. Sharlet, J. Ragan-Kelley, and K. Fatahalian, "Automatically scheduling halide image processing pipelines," *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016, ISSN: 0730-0301. DOI: 10.1145/2897824.2925952. [Online]. Available: https://doi.org/10.1145/2897824.2925952.

[12] J. Won, C. Mendis, J. S. Emer, and S. Amarasinghe, "Waco: Learning workload-aware co-optimization of the format and schedule of a sparse tensor program," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023, Vancouver, BC, Canada: Association for Computing Machinery, 2023, pp. 920–934, ISBN: 9781450399166. DOI: 10.1145/3575693.3575742. [Online]. Available: https://doi.org/10.1145/3575693.3575742.

[13] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, "Taichi: A language for high-performance computation on spatially sparse data structures," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–16, 2019.

[14] Z. DeVito, M. Mara, M. Zollhöfer, G. Bernstein, J. Ragan-Kelley, C. Theobalt, P. Hanrahan, M. Fisher, and M. Niessner, "Opt: A domain specific language for non-linear least squares optimization in graphics and imaging," *ACM Transactions on Graphics (TOG)*, vol. 36, no. 5, pp. 1–27, 2017.

[15] Y. Zhang, M. Yang, R. Baghdadi, S. Kamil, J. Shun, and S. Amarasinghe, "Graphit: A high-performance graph dsl," *Proceedings of the ACM on Programming Languages*, vol. 2, no. OOPSLA, pp. 1–30, 2018.

[16] B. Hagedorn, J. Lenfers, T. Kœhler, X. Qin, S. Gorlatch, and M. Steuwer, "Achieving high-performance the functional way: A functional pearl on expressing high-performance optimizations as rewrite strategies," en, *Proceedings of the ACM on Programming Languages*, vol. 4, no. ICFP, pp. 1–29, Aug. 2020, ISSN: 2475-1421. DOI: 10.1145/3408974. [Online]. Available: https://dl.acm.org/doi/10.1145/3408974 (visited on 04/29/2022).

[17] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: End-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, vol. 11, no. 2018, p. 20, 2018.

[18] A. Liu, G. L. Bernstein, A. Chlipala, and J. Ragan-Kelley, "Verified tensor-program optimization via high-level scheduling rewrites," *Proceedings of the ACM on Programming Languages*, vol. 6, no. POPL, 55:1–55:28, Jan. 2022. DOI: 10.1145/3498717. [Online]. Available: https://dl.acm.org/doi/10.1145/3498717 (visited on 04/03/2024).

[19] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, and J. M. Moura, "Spiral: Extreme

performance portability," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1935–1968, 2018.