Shriram Krishnamurthi
Martin Odersky (Eds.)

LNCS 4420

Compiler Construction

16th International Conference, CC 2007
Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007
Braga, Portugal, March 2007, Proceedings

European Joint Conferences on
Theory
And
Practice of
Software
2007

Springer

Shriram Krishnamurthi   Martin Odersky (Eds.)

Compiler Construction

16th International Conference, CC 2007
Held as Part of the Joint European Conferences
on Theory and Practice of Software, ETAPS 2007
Braga, Portugal, March 26-30, 2007
Proceedings

Springer

Volume Editors

Shriram Krishnamurthi
Brown University
Computer Science Department
Providence, RI, USA
E-mail: sk@cs.brown.edu

Martin Odersky
EPFL
School of Computer and Communication Sciences
Institute of Core Computing
Lausanne, Switzerland
E-mail: martin.odersky@epfl.ch

Library of Congress Control Number: 2007921957

CR Subject Classification (1998): D.3.4, D.3.1, F.4.2, D.2.6, F.3, I.2.2

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-71228-3 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-71228-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12029303 06/3142 5 4 3 2 1 0

Foreword

ETAPS 2007 is the tenth instance of the European Joint Conferences on Theory and Practice of Software, and thus a cause for celebration.

The events that comprise ETAPS address various aspects of the system development process, including specification, design, implementation, analysis and improvement. The languages, methodologies and tools which support these activities are all well within its scope. Different blends of theory and practice are represented, with an inclination towards theory with a practical motivation on the one hand and soundly based practice on the other. Many of the issues involved in software design apply to systems in general, including hardware systems, and the emphasis on software is not intended to be exclusive.

History and Prehistory of ETAPS

ETAPS as we know it is an annual federated conference that was established in 1998 by combining five conferences [Compiler Construction (CC), European Symposium on Programming (ESOP), Fundamental Approaches to Software Engineering (FASE), Foundations of Software Science and Computation Structures (FOSSACS), Tools and Algorithms for Construction and Analysis of Systems (TACAS)] with satellite events.

All five conferences had previously existed in some form and in various colocated combinations: accordingly, the prehistory of ETAPS is complex. FOSSACS was earlier known as the Colloquium on Trees in Algebra and Programming (CAAP), being renamed for inclusion in ETAPS as its historical name no longer reflected its contents. Indeed CAAP's history goes back a long way; prior to 1981, it was known as the Colleque de Lille sur les Arbres en Algebre et en Programmation. FASE was the indirect successor of a 1985 event known as Colloquium on Software Engineering (CSE), which together with CAAP formed a joint event called TAPSOFT in odd-numbered years. Instances of TAPSOFT, all including CAAP plus at least one software engineering event, took place every two years from 1985 to 1997 inclusive. In the alternate years, CAAP took place separately from TAPSOFT.

Meanwhile, ESOP and CC were each taking place every two years from 1986. From 1988, CAAP was colocated with ESOP in even years. In 1994, CC became a "conference" rather than a "workshop" and CAAP, CC and ESOP were thereafter all colocated in even years.

TACAS, the youngest of the ETAPS conferences, was founded as an international workshop in 1995; in its first year, it was colocated with TAPSOFT. It took place each year, and became a "conference" when it formed part of ETAPS 1998. It is a telling indication of the importance of tools in the modern field of informatics that TACAS today is the largest of the ETAPS conferences.

The coming together of these five conferences was due to the vision of a small group of people who saw the potential of a combined event to be more than the sum of its parts. Under the leadership of Don Sannella, who became the first ETAPS steering committee chair, they included: Andre Arnold, Egidio Astesiano, Hartmut Ehrig, Peter Fritzson, Marie-Claude Gaudel, Tibor Gyimothy, Paul Klint, Kim Guldstrand Larsen, Peter Mosses, Alan Mycroft, Hanne Riis Nielson, Maurice Nivat, Fernando Orejas, Bernhard Steffen, Wolfgang Thomas and (alphabetically last but in fact one of the ringleaders) Reinhard Wilhelm.

ETAPS today is a loose confederation in which each event retains its own identity, with a separate programme committee and proceedings. Its format is open-ended, allowing it to grow and evolve as time goes by. Contributed talks and system demonstrations are in synchronized parallel sessions, with invited lectures in plenary sessions. Two of the invited lectures are reserved for "unifying" talks on topics of interest to the whole range of ETAPS attendees. The aim of cramming all this activity into a single one-week meeting is to create a strong magnet for academic and industrial researchers working on topics within its scope, giving them the opportunity to learn about research in related areas, and thereby to foster new and existing links between work in areas that were formerly addressed in separate meetings.

ETAPS 1998–2006

The first ETAPS took place in Lisbon in 1998. Subsequently it visited Amsterdam, Berlin, Genova, Grenoble, Warsaw, Barcelona, Edinburgh and Vienna before arriving in Braga this year. During that time it has become established as the major conference in its field, attracting participants and authors from all over the world. The number of submissions has more than doubled, and the numbers of satellite events and attendees have also increased dramatically.

ETAPS 2007

ETAPS 2007 comprises five conferences (CC, ESOP, FASE, FOSSACS, TACAS), 18 satellite workshops (ACCAT, AVIS, Bytecode, COCV, FESCA, FinCo, GT-VMT, HAV, HFL, LDTA, MBT, MOMPES, OpenCert, QAPL, SC, SLA++P, TERMGRAPH and WITS), three tutorials, and seven invited lectures (not including those that were specific to the satellite events). We received around 630 submissions to the five conferences this year, giving an overall acceptance rate of 25%. To accommodate the unprecedented quantity and quality of submissions, we have four-way parallelism between the main conferences on Wednesday for the first time. Congratulations to all the authors who made it to the final programme! I hope that most of the other authors still found a way of participating in this exciting event and I hope you will continue submitting.

ETAPS 2007 was organized by the Departamento de Informática of the Universidade do Minho, in cooperation with

- European Association for Theoretical Computer Science (EATCS)
- European Association for Programming Languages and Systems (EAPLS)
- European Association of Software Science and Technology (EASST)
- The Computer Science and Technology Center (CCTC, Universidade do Minho)
- Camara Municipal de Braga
- CeSIUM/GEMCC (Student Groups)

The organizing team comprised:

- João Saraiva (Chair)
- José Bacelar Almeida (Web site)
- José João Almeida (Publicity)
- Luís Soares Barbosa (Satellite Events, Finances)
- Victor Francisco Fonte (Web site)
- Pedro Henriques (Local Arrangements)
- José Nuno Oliveira (Industrial Liaison)
- Jorge Sousa Pinto (Publicity)
- António Nestor Ribeiro (Fundraising)
- Joost Visser (Satellite Events)

ETAPS 2007 received generous sponsorship from Fundação para a Ciência e a Tecnologia (FCT), Enabler (a Wipro Company), Cisco and TAP Air Portugal.

Overall planning for ETAPS conferences is the responsibility of its Steering Committee, whose current membership is:

Perdita Stevens (Edinburgh, Chair), Roberto Amadio (Paris), Luciano Baresi (Milan), Sophia Drossopoulou (London), Matt Dwyer (Nebraska), Hartmut Ehrig (Berlin), José Fiadeiro (Leicester), Chris Hankin (London), Laurie Hendren (McGill), Mike Hinchey (NASA Goddard), Michael Huth (London), Anna Ingólfsdóttir (Aalborg), Paola Inverardi (L'Aquila), Joost-Pieter Katoen (Aachen), Paul Klint (Amsterdam), Jens Knoop (Vienna), Shriram Krishnamurthi (Brown), Kim Larsen (Aalborg), Tiziana Margaria (Göttingen), Ugo Montanari (Pisa), Rocco de Nicola (Florence), Jakob Rehof (Dortmund), Don Sannella (Edinburgh), João Saraiva (Minho), Vladimiro Sassone (Southampton), Helmut Seidl (Munich), Daniel Varro (Budapest), Andreas Zeller (Saarbrücken).

I would like to express my sincere gratitude to all of these people and organizations, the programme committee chairs and PC members of the ETAPS conferences, the organizers of the satellite events, the speakers themselves, the many reviewers, and Springer for agreeing to publish the ETAPS proceedings. Finally, I would like to thank the organizing chair of ETAPS 2007, João Saraiva, for arranging for us to have ETAPS in the ancient city of Braga.

Edinburgh, January 2007

Perdita Stevens
ETAPS Steering Committee Chair

Preface

This volume constitutes the proceedings of the 2007 Compiler Construction (CC) conference, held March 26–27, 2007 in Braga, Portugal as part of the ETAPS umbrella.

In keeping with tradition, CC solicited both research descriptions and tool papers, though most submissions were in the former category. We accepted 14 out of 60 submissions, all in the research paper category. Each submission was reviewed by at least three PC members, with papers co-authored by PC members receiving at least one additional reviewer. Forty-two papers had at least some support, and thus warranted deliberation. (Only one paper was entirely outside the scope of the conference.) The papers were discussed at a (lively!) live PC meeting held in Lausanne, Switzerland, on December 4, 2006. Almost all PC members attended the meeting, with the remainder participating by telephone.

CC had, in a few instances, to contend with the growing problem of defining what constitutes a prior publication in an era when workshops are now published in the ACM's Digital Library (and other such formal on-line repositories). As we have observed on the PCs of other venues, PC members in CC had mixed views on this matter. In a few years we therefore expect to see standardized policy to cover such cases.

We thank the many people who eased the administration of CC 2007. Foremost, the PC (and their subreviewers) did a thorough and conscientious job. Perdita Stevens invested enormous effort into the smooth running of ETAPS. Jay McCarthy offered round-the-clock support for the Continue software that handled our papers. Yvette Dubuis (of EPFL) and Dawn Reed (of Brown) provided stellar administrative support to the Chairs, with Mme. Dubuis also organizing the PC meeting. EPFL helped sponsor the PC meeting. Philipp Haller provided valuable help in preparing the proceedings. Finally, the ETAPS and CC Steering Committees ensured the smooth passage of many matters.

January 2007

Shriram Krishnamurthi
Martin Odersky

Organization

Program Committee

Chairs: Shriram Krishnamurthi, Brown University, Providence
Martin Odersky, EPFL, Lausanne

Eric Allen, Sun Microsystems, Inc.
Emery Berger, University of Massachusetts Amherst
Rastislav Bodik, University of California, Berkeley
William Cook, University of Texas at Austin
Chen Ding, University of Rochester
Sabine Glesner, Technical University of Berlin
Dan Grossman, University of Washington
Rajiv Gupta, University of Arizona
Andrew Kennedy, Microsoft Research Cambridge
Christian Lengauer, University of Passau
Cristina Videira Lopes, University of California, Irvine
Todd Millstein, University of California, Los Angeles
G. Ramalingam, Microsoft Research India
Vijay Saraswat, IBM TJ Watson Research Center
Zhong Shao, Yale University
Yannis Smaragdakis, University of Oregon
Gregor Snelting, University of Passau
Joost Visser, Universidade do Minho
Reinhard Wilhelm, Saarland University

Reviewers

Arnold, Gilad
Arnold, Matthew
Barik, Raj
Bastoul, Cedric
Bierman, Gavin
Bond, Michael
Brandes, Thomas
Chilimbi, Trishul
Cohen, Albert
Felleisen, Matthias
Größlinger, Armin
Griebl, Martin
Grund, Daniel
Lin, Calvin
Lucas, Philipp
Marlow, Simon
Merz, Peter
Naeem, Nomair
Nagarajan, Vijay
Palsberg, Jens
Penso, Lucia Draque
Pierce, Benjamin
Pister, Markus
Rabbah, Rodric
Reineke, Jan
Reppy, John

Herrmann, Christoph A.	Shankar, AJ
Hind, Mike	Solar-Lezama, Armando
Ibrahim, Ali	Sridharan, Manu
Joyner, Mackale	Tallam, Sriraman
Jump, Maria	Vouillon, Jerome
Kelsey, Kirk	Whalley, David
Kitchin, David	Wiedermann, Benjamin
Kulkarni, Prasad	Ylvisaker, Benjamin
Lhotak, Ondrej	von Praun, Christoph

Table of Contents

New Algorithms for SIMD Alignment*

Liza Fireman[1], Erez Petrank[2,**], and Ayal Zaks[3]

[1] Dept. of Computer Science, Technion, Haifa 32000, Israel
liza@cs.technion.ac.il
[2] Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA
erez@cs.technion.ac.il
[3] IBM Haifa Research Laboratory, Mount Carmel, Haifa 31905, Israel
zaks@il.ibm.com

Abstract. Optimizing programs for modern multiprocessor or vector platforms is a major important challenge for compilers today. In this work, we focus on one challenging aspect: the SIMD ALIGNMENT problem. Previously, only heuristics were used to solve this problem, without guarantees on the number of shifts in the obtained solution. We study two interesting and realistic special cases of the SIMD ALIGNMENT problem and present two novel and efficient algorithms that provide *optimal* solutions for these two cases. The new algorithms employ dynamic programming and a MIN-CUT/MAX-FLOW algorithm as subroutines. We also discuss the relation between the SIMD ALIGNMENT problem and the MULTIWAY CUT and NODE MULTIWAY CUT problems; and we show how to derive an approximated solution to the SIMD ALIGNMENT problem based on approximation algorithms to these two known problems.

1 Introduction

Designing effective optimizations for modern architectures is an important goal for compiler designers today. This general task is composed of many non-trivial problems, the solution to which is not always known. In this paper we study one such problem — the SIMD ALIGNMENT problem, which emerges when optimizing for multimedia extensions. Previously only heuristics were studied for this problem [25,30,14]. In this paper we present two novel algorithms that obtain optimal solutions for two special cases. These special cases are actually broad enough to cover many practical instances of the SIMD ALIGNMENT problem.

Multimedia extensions have become one of the most popular additions to general-purpose microprocessors. Existing multimedia extensions are characterized as Single Instruction Multiple Data (SIMD) units that support packed, fixed-length vectors, such as MMX and SSE for Intel and AltiVec for IBM, Apple and Motorola. Producing SIMD codes is sometimes done manually for important specific application, but is often produced automatically by compilers (referred to as auto-vectorization or simdization). Explicit vector programming is time consuming and error prone. A promising alternative is to exploit vectorization technology to automatically generate SIMD codes from

* This research was supported by THE ISRAEL SCIENCE FOUNDATION (grant No. 845/06).
** On sabbatical leave from the Computer Science Department, Technion, Haifa 32000, Israel.

programs written in standard high-level languages. However, simdization is not trivial. Some of the difficulties in optimizing code for SIMD architectures stem from hardware constraints imposed by today's SIMD architectures [25].

One restrictive hardware feature that can significantly impact the effectiveness of simdization is the alignment constraint of memory units. In AltiVec [10], for example, a load instruction loads 16-byte contiguous memory from 16-byte aligned memory address (by ignoring the least significant 4 bits of the given memory address). The same applies to store instructions. Now consider a stream: given a stride-one memory reference in a loop, a *memory stream* corresponds to all the contiguous locations in memory accessed by that memory reference over the lifetime of the loop. The alignment constraint of SIMD memory units requires that streams involved in the same SIMD operation must have matching offsets.

Consider the following code fragment, where integer arrays a, b, and c start at 16-byte aligned addresses.

```
for ($i$ = 0; $i$ < 1000; $i$++) do
    $a[i] = b[i+1] + c[i+2];$
end for
```

The above code includes a loop with misaligned references. It requires additional realignment operations to allow vectorization on SIMD architectures with alignment constraints. In particular, unless special care is taken, data involved in the same computation, i.e., $a[i]$, $b[i+1]$, $c[i+2]$, will be relatively misaligned after being loaded to machine registers. To produce correct results, this data must be reorganized to reside in the same slots of their corresponding registers prior to performing any arithmetic computation.

The realigning of data in registers is achieved by explicit shift instructions that execute inside the loop and therefore affect performance significantly. The problem is to automatically reorganize data streams in registers to satisfy the alignment requirements imposed by the hardware using a minimum number of shift executions. Prior research has focused primarily on vectorizing loops where all memory references are properly aligned. An important aspect of this problem, namely, the problem of minimizing the number of shifts for aligning a given expression has been studied only recently.

An alternative to performing shift operations at runtime is to modify the layout of the data in memory. This alternative suffers from several obvious limitations. In this paper the initial data alignment is assumed to be predetermined.

1.1 This Work

In this work we investigate the computational complexity of the SIMD ALIGNMENT problem. A formal definition of the problem, motivation, and examples from current modern platforms appear in Section 2. The main contribution of this paper is the presentation of two new algorithms that provide *optimal* solutions for two special cases. These special cases are quite general and cover many practical instances.

A polynomial-time algorithm for expressions with two alignments. For expressions that contain two distinct predetermined alignments, an efficient algorithm based on solving a MINIMUM NODE S-T CUT problem can compute an optimal solution to the SIMD ALIGNMENT problem.

A polynomial-time algorithm for a single-appearance tree expression. For expressions that contain no common sub-expressions (i.e. form a tree) and where each array appears only once in the expression, an efficient algorithm based on dynamic programming can compute an optimal solution to the SIMD ALIGNMENT problem.

We stress that both cases are realistic and are common in practice. Also, the two cases do not supersede one another: one case is broader in the sense that it works on any expression, not necessarily a tree, and the other case is broader in the sense that it applies to an arbitrary number of alignments.

The SIMD ALIGNMENT problem can be mapped to the MULTIWAY CUT problem, and known algorithms for MULTIWAY CUT [11] can be used to solve the SIMD ALIGNMENT problem. However, known hardness results [12,5] do not hold for the SIMD ALIGNMENT problem if a shifted stream may be used in both its original form and its shifted form (see Section 9). Furthermore, the mapping to the MULTIWAY CUT problem is more involved in this case (see Section 7).

1.2 Organization

In Section 2 we formally define the SIMD ALIGNMENT problem. In Section 3 we list useful heuristics proposed in the literature so far.In Section 4 we propose a graph representation of the SIMD ALIGNMENT problem, which will be used by the algorithms. In Sections 6 and 5 we present the efficient algorithms for the special case of expressions with only two alignments and for single-appearance tree expressions, respectively, and in Section 8 we present the effectiveness of these algorithms. Section 7 relates the SIMD ALIGNMENT problem to MULTIWAY CUT problems. Relevant prior art is described in Section 9 and Section 10 concludes.

2 An Overview of the SIMD ALIGNMENT Problem

We begin by defining the SIMD ALIGNMENT problem.

Definition 1. The SIMD ALIGNMENT **problem**
Input: *An expression containing input operands, sub-expressions (operations) and output operands, with an alignment value assigned to every input and output operand.*
Solution: *A specification of shifts for some input operands and operations, such that the inputs to each operation all have the same alignment values and the inputs to output operands have the desired alignment values.*
Cost: *The number of shifts in the solution.*

Note that for each operation of the given expression, the solution may specify several shifts if the result of the operation is needed in different alignments, or it may specify no shifts at all if the result is needed only in the same alignment as its inputs. This applies to input operands as well. However, a solution is feasible only if the inputs of *each* operation and output operand are properly aligned. An elaborate description of how the shift operation is used with real platforms and a detailed example may be found in the thesis [15].

Shifting of data from one alignment value to another requires one shift operation but typically may require preliminary preparation of pre-loading and setting shift amount [23]. This preliminary work can often be placed before the loop where it is tolerable, whereas the shift operations themselves are part of the expression and must remain inside the loop. That is why our objective is to minimize the number of shifts.

3 Previous Heuristics

Previous work concentrated on identifying operations that can be vectorized assuming all operands are aligned. Several simple heuristics have been proposed to solve the alignment problem. In this section we shortly survey these heuristics, originally presented in [14], each of which can be shown to be sub-optimal for simple realistic instances [15].

- *Zero-Shift Policy.* This policy shifts each misaligned load stream to offset zero, and shifts the store stream from offset zero to the alignment of the store address. This simple policy is employed by the widespread GCC compiler [21,24].
- *Eager-Shift Policy.* This policy shifts each load stream to the alignment of the store.
- *Lazy-Shift Policy.* This is a greedy policy of inserting shifts as late as possible in the expression. This policy does not specify how to break ties when different shifts may be used.
- *Majority Policy.* This policy shifts each load stream to the majority of the alignments of the input and output streams, and shifts the store stream from the majority offset to the alignment of the store address.

4 An Abstraction of SIMD Alignment

In what follows, it will be useful to represent instances of the SIMD ALIGNMENT problem using annotated graphs. We provide a graph representation for instances in which an array appears in the expression in one alignment only. The proposed representation may also be used in the general case, but for arrays appearing with multiple alignments the cost of the solution cannot be easily translated from graphs to expressions. This is because a single shift can be used for multiple alignments of the same array (e.g. to align both $a[i]$ to match $b[i+1]$ and $a[i+1]$ to match $c[i+2]$). Note, however, that this does not hold if only two distinct alignments are considered, as is the case in Section 6.

We call an expression in which each array appears with one alignment only a *single-appearance expression*. Most of the techniques employed in this paper relate to the study of graph algorithms. The representation of a single-appearance expression as a directed graph is the standard representation of expressions as graphs, except for two modifications. First, we add alignment labels to the nodes. Second, all appearances of the same array are represented by a single node. The nodes that represent the input and output streams are associated with alignment labels that signify the initial and final alignments, respectively, and the name of the array. Each operation (sub-expression) is also represented by a graph node. The operation nodes are labelled with the operation they carry. The nodes for input streams of an operation are connected to the node of the operation by incoming edges.

Consider the following example:

```
for ($i = 0$; $i < 1000$; $i$++) do
    $a[i+3] = b[i+1] * c[i+2] + c[i+2] * d[i+1]$;
end for
```

The corresponding graph representation for the above expression is shown in Figure 1. This graph has three leaves (input nodes) labelled $1(b), 2(c), 1(d)$ and one root (output node) labelled $3(a)$. The labels signify the initial and final alignments and the array names. The operation nodes are labelled with the operation they represent. Note that the graph in this example in not a tree. It is a Directed Acyclic Graph (DAG).

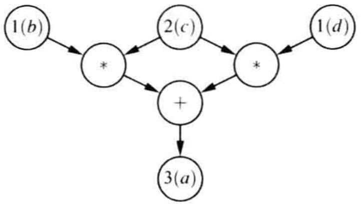

Fig. 1. A graph representation of $a[i+3] = b[i+1] * c[i+2] + c[i+2] * d[i+1]$

4.1 A Solution to a Graph Representation of a Single-Appearance

An important property of the expression execution is that once we shift a stream, we can use the shifted stream repeatedly without paying more shifts. In addition, even if we shift a stream we can still use its original alignment. We consider this property in the solution representation and its cost definition.

A solution to the graph representation of a single-appearance SIMD ALIGNMENT problem is a labelling of the nodes. The cost of a solution for the graph $G(V,E)$ is the sum of the costs $c(v)$ associated with each node $v \in V$, where $c(v)$ is the number of distinct labels of v's successor nodes that are also different from the label of v. We claim that this cost of the graph solution is equal to the cost of the corresponding solution of the expression. We interpret the solution to the graph as shifting specifications for the expression execution as follows. Each operation is executed at the alignment that is the label of its corresponding node in the graph. A stream represented by node v should be shifted from the alignment represented by its label to the alignments of its successors (if different from its own). This specifies a valid execution of the expression because all operations have their input stream shifted to the same alignment. We need to show that the computed cost represents the minimal number of shifts required to execute the operations at the alignment specified by the graph solution. The expression is a single-appearance expression and therefore a shift must be done for a node if its successors do not have the same label. If an array appears with more than one alignment in the expression, shifting it once could save shift to another use of this array. We do not deal with sharing shifts among multiple alignment appearances of input operands or subexpressions. Therefore, the cost of the solution for a graph is exactly the number of shifts that should be executed in order to compute the expression with the alignments specified by the graph solution. In Figure 2 we show an example of a graph with a given solution.

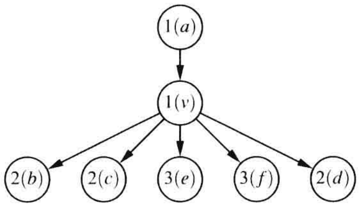

Fig. 2. A graph that exemplifies the cost of SIMD

The labelling shown in Figure 2 costs only two shifts, because the descendants of v have only two distinct shift labels that are also different from its own label (2 and 3). Therefore, v should be shifted from alignment 1 to alignments 2 and 3, enabling the execution of the rest of the computation without any further shift.

We are now ready to define the problem SIMDG.

Definition 2 (The SIMDG Problem)
Input: *(G,L) where $G(V,E)$ is a DAG representation of a single-appearance expression and L is a set of predetermined shift labels for the source and sink nodes.*
Solution: *a labelling c for all nodes, which is an extension of the given labelling L.*
Cost function: *for a labelling c the cost is:*

$$\sum_{v\in V} |\{c(u) \ : \ \exists u \in S(v) \ \ c(u) \neq c(v)\}|$$

where $S(v)$ is the set of successor nodes of v.
Goal: *finding a solution with minimum cost.*

From this point on we stick to the graph representation and consider the SIMDG problem rather than the original SIMD ALIGNMENT problem.

5 A Polynomial-Time Algorithm for Single-Appearance Tree Expressions

In this section we deal with expressions having an arbitrary number of alignments, but whose graph representations form a tree and any array appears in the expression at most once. We show that the SIMD ALIGNMENT problem can be solved in polynomial-time using dynamic programming in such cases.