Leibniz Universität Hannover
Fakultät für Mathematik und Physik
Institut für Algebraische Geometrie

in Kooperation mit
Technische Universität Kaiserslautern
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik

Computation of the GIT-fan using a massively parallel implementation

Masterarbeit

Hannover, 11.01.2018
Christian Reinbold
Matrikelnr.: 2942360

Erstprüfer:	apl. Prof. Dr. Anne Frühbis-Krüger
Zweitprüfer:	Dr. Janko Böhm
Betreuer:	apl. Prof. Dr. Anne Frühbis-Krüger
	Dr. Janko Böhm
	Dr. Mirko Rahn

Abstract

Fancy, english text.

Kurzzusammenfassung

Brillierender, deutscher Text.

Acknowledgements

Syn, Ack...

Contents

1 Introduction

Well, skip this and just read the rest. Less work for me.

2 Preliminaries

2.1 algebraic group actions

Torus as special case

2.2 Categorical quotients

2.3 GIT quotients & fans

Reduction of arbitrary groups to the torus case

2.4 Mori dream spaces, movable divisor classes?

3 Concept of the algorithm

3.1 Computing orbit cones

Monomial containment test Moving Cone

3.2 Traversing the GIT fan

3.3 Exploiting symmetry

4 Implementation

In the following we present the implementation of the algorithm discussed in the previous chapter. In contrast to the existing SINGULAR implementation *gitfan.lib* [2], which forms the foundation of our work, the approach described here utilises high performance computing methods in order to speed up the execution by running the algorithm on any number of machines simultaneously. Naturally, one has to identify independent components permitting concurrent execution on the one hand and inherent sequential processes on the other hand. As the sequential parts of the algorithm mostly agree with *gitfan.lib* – only signatures have been modified, providing an appropriate interface for the superordinate parallel environment – we skip a detailed discussion of those and refer the interested reader to [2]. The chapter is structured as follows: First, we discuss the employed parallelisation framework GPI-SPACE developed by the *Fraunhofer-Institut für Techno- und Wirtschaftsmathematik* (Fraunhofer ITWM) and follow up with the integration of singular code into GPI-SPACE applications. Next, a parallel design of algorithm (ref needed to main algorithm in previous chapter) is presented. A separate section covers approaches for managing found GIT cones, a potential bottleneck due to data dependencies. Finally, we describe input and output formats, additional features such as speeding up the execution by precomputed results and executed software tests in order to verify the functional correctness of the program.

4.1 GPI-SPACE

GPI-SPACE is a workflow management system developed by the Fraunhofer ITWM which supports the execution of arbitrary workflows on ultra scale systems [5]. It consists of three key components:

- the distributed run-time engine (DRTS), which is responsible for building and managing arbitrary worker topologies based on the available compute nodes, resource management and job scheduling. It supports the reallocation of jobs in case of hardware failures and further integrates dynamically added hardware

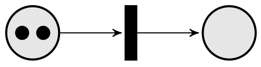

Figure 1: Graphical representation of a Petri net with two places and one transition. The current state is a marking that sends the left place to 2 and the second place to 0.

- the workflow engine (WE). It tracks the state of the workflow, identifies a front of activities for which all dependencies are resolved and laces jobs from input data and active transitions which then are sent to the DRTS for scheduling.
- a virtual memory layer, which provides the necessary infrastructure for the partitioned global address space (PGAS) programming model. It relies on GPI by Fraunhofer ITWM [4].

The framework has been developed with separation of concerns in mind, whereby the concerns here are given by computation and coordination [3]. In our case the computation takes place in sequential Singular-routines. The dependencies and data transfers between this routines, which assemble the particular routines into a complex, parallel algorithm, are described by a special domain language in the the coordination layer. In GPI-Space, this domain language is chosen to be an extension of the well known Petri net model introduced in 1962 by Carl Adam Petri [7].

Petri nets

Formally, an (unweighted) *Petri net* is a triple (P,T,F) of *places* P, *transitions* T and directed *arcs* $F \subseteq (P \times T) \cup (T \times P)$ relating the former concepts. We demand that $P \cap T = \emptyset$. A function $M\colon P \to \mathbb{N}$ is called *marking* and describes a possible state of the Petri net. If we have $M(p) = k$ for a place p and the current state of the Petri net is given by M, we say that p holds k *tokens*. Visually, circles depict places, rectangles are transitions and arrows between circles and rectangles represent arcs whose direction is indicated by the tip. The current state M of the Petri net is displayed by placing $M(p)$ dots in the circle representing the place p. Figure 1 gives an example for the graphical representation of a Petri net.

A transition t is said to be *active*, iff

$$\forall p \in P\colon (p,t) \in F \Rightarrow M(p) > 0$$

holds. In this case *firing* t means to transform the current state M into M' by the

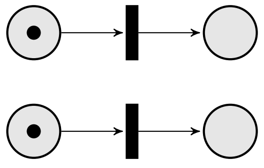

Figure 2: Modelling concurrent processes in a Petri net.

update rule

$$M'(p) := \begin{cases} M(p) - 1 & (p,t) \in F,\ (t,p) \notin F \\ M(p) + 1 & (p,t) \notin F,\ (t,p) \in F \\ M(p) & \text{else} \end{cases}$$

for all $p \in P$. We interpret this as follows: For every incoming arc (p,t) from p a token in p is consumed and for every outgoing arc (t,p') to p' a new token is placed into p'. Note that the notion of tokens "moving through transitions" instead of consumption and creation of tokens is erroneous as we will see later on when attaching data to them.

An important property of Petri nets is given by the fact that in many cases the same result is obtained if the order in which two transitions t_1 and t_2 fire is reversed. This allows the modelling of concurrent behaviour. When executing the Petri net model on a machine, the necessary steps for firing t_1 and t_2 respectively may be performed in parallel. Figure 2 depicts such a situation. Note that figure 1 also shows an example for concurrent behaviour in which the transition may fire twice. However, the order in which the tokens in the left place are consumed is of no relevance. This example seems trivial as tokens located at the same place are indistinguishable. However, this is not the case anymore when attaching data to tokens. In fact, figure 1 depicts the common scenario of concurrent behaviour in our algorithm, as parallel execution is mostly achieved by invoking the same routine for varying, independent input data.

Coloured Petri nets and guards

Workflows describe a (possible concurrent) progression of processes, hence it is evident to model a process by a transition. The input and output data of a process is taken into account by attaching data values of a predefined type to each token. Then every incoming arc provides input data for the process corresponding to the target transition, whereas every outgoing arc describes output data of the source transition that should be put into the target place. A place may only hold tokens of a single type, introducing strong typing to Petri nets. This kind of Petri net is

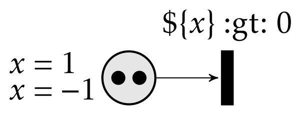

Figure 3: Guards in a Petri net. Only the token with $x = 1$ activates the transition.

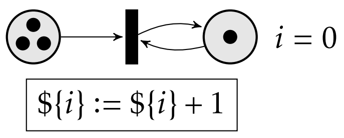

Figure 4: Expressions in a Petri net. After firing the transition three times, the left place contains no tokens anymore and the right place contains one token with $i = 3$.

called *Coloured Petri net*, developed by Kurt Jensen in 1980. The term originates from the analogy of understanding each distinct type as a unique color. A formal treatment of this concept may be found in [6].

In order to control the data flow, each transition may be tagged with a predicate over the types of places with incoming arcs, which is called *condition* or *guard* [6]. The activation rule for a transition is extended such that not only a token has to be present for every incoming arc, but also the data values of the consumed tokens have to satisfy the condition. GPI-Space provides its own expression language in order to specify conditions. Figure 3 shows a transition that consumes tokens with positive data values only.

Expressions

GPI-Space supports two kinds of transitions. The first kind triggers a module call whenever firing, executing arbitrary C++-code where the input and output tokens are mapped to references. The module call is packaged as a job and scheduled by the DRTS. A lightweight approach that requires no scheduling is given by the second kind of transition. The expression language provided by GPI-Space allows to specify the output data values of a transition by means of simple operations on the input data values. This extension is intended to be used for minor computations and saves the overhead of generating and scheduling a job by executing the expression directly within the WE. For instance, counting consumed tokens may be realised that way, see figure 4. If the counting would be realised with C++, each increment will cause a disproportionate communication and scheduling overhead.

[1]

4.2 Integration of Singular

drawbacks

4.3 Application flow & Concurrency

4.4 GIT cone data structure

4.5 Input & Output structure

4.6 Additional features

an Kommandozeilenparameter entlanghangeln

4.7 Software testing

5 Applications

5.1 Grassmanian $\mathbb{G}(2,5)$

5.2 Moduli space of stable curves of genus 0

6 Conclusion

Let $p = x_1x_3 + x_2x_4 + x_5 \in \mathbb{K}[x_1, x_2, x_3, x_4, x_5] =: \mathbb{K}[\mathbf{x}]$. Set

$$\mathfrak{a} := \langle x_3, x_4 \rangle \cdot \langle p \rangle,$$

$$Q = (q_1, q_2, q_3, q_4, q_5) := \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{pmatrix}.$$

Note that Q defines a $\mathbb{Z}^3$ grading on $\mathbb{K}[x_1, x_2, x_3, x_4, x_5]/_{\mathfrak{a}}$ since p and therefore $\mathfrak{a}$ is homogenous with respect to the grading $\deg(x_i) = q_i$. Thus, we obtain a torus action of $(\mathbb{K}^*)^3$ on $V(\mathfrak{a})$.

The $\mathfrak{a}$-faces with full dimensional image in $Q(\gamma)$, $\gamma = \mathbb{Q}^5_{\geq 0}$, are given by

$$\begin{aligned} \gamma_1 &:= \langle e_1, e_2, e_5 \rangle \\ \gamma_2 &:= \langle e_1, e_2, e_3, e_5 \rangle \\ \gamma_3 &:= \langle e_1, e_2, e_4, e_5 \rangle \\ \gamma_4 &:= \langle e_1, e_3, e_4, e_5 \rangle \\ \gamma_5 &:= \langle e_2, e_3, e_4, e_5 \rangle \\ \gamma_6 &:= \langle e_1, e_2, e_3, e_4 \rangle \\ \gamma_7 &:= \langle e_1, e_2, e_3, e_4, e_5 \rangle \end{aligned}$$

This is easily seen by sending all variables x_i with $e_i \notin \gamma_j$ in $\mathfrak{a}$ to zero and check that the new ideal does not contain any monomials. (details see below)

The orbit cones $Q(\gamma_j) \in \mathbb{Q}^3_{z_1,z_2,z_3}$ have the following form when intersecting them with the $\{z_3 = 1\}$-plain (vice versa, the orbit cones are recovered by expanding the polytopes in z_3-direction to cones in $\mathbb{Q}^3$):

A Something appendixable

Theorem A.1 (Great theorem) *Problem in appendix.*

Proof: by intimidation. □

Bibliography

[1] W. M. P. van der Aalst. 'Three Good Reasons for Using a Petri-Net-Based Workflow Management System'. In: *Information and Process Integration in Enterprises: Rethinking Documents*. Ed. by Toshiro Wakayama, Srikanth Kannapan, Chan Meng Khoong, Shamkant Navathe, and JoAnne Yates. Boston, MA: Springer US, 1998, pp. 161–182. ISBN: 978-1-4615-5499-8. DOI: 10.1007/978-1-4615-5499-8_10.

[2] Janko Böhm, Wolfram Decker, Simon Keicher, and You Ren. *gitfan.lib – A Singular library for computing the GIT fan*. 2016. URL: https://github.com/Singular/Sources (visited on 10/16/2017).

[3] David Gelernter and Nicholas Carriero. 'Coordination Languages and Their Significance'. In: *Commun. ACM* 35.2 (Feb. 1992), pp. 96–107. ISSN: 0001-0782. DOI: 10.1145/129630.376083.

[4] *GPI*. URL: http://www.gpi-site.com/gpi2/ (visited on 10/16/2017).

[5] *GPI-Space*. URL: http://www.gpi-space.de/ (visited on 10/16/2017).

[6] Kurt Jensen. 'Coloured Petri nets'. In: *Petri Nets: Central Models and Their Properties*. Ed. by W. Brauer, W. Reisig, and G. Rozenberg. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1987, pp. 248–299. ISBN: 978-3-540-47919-2. DOI: 10.1007/BFb0046842.

[7] Carl Adam Petri. 'Kommunikation mit Automaten'. PhD thesis. Institut für Angewandte Mathematik der Universität Bonn, 1962.

List of abbreviations

DRTS	distributed run-time engine	7
Fraunhofer ITWM	*Fraunhofer-Institut für Techno- und Wirtschaftsmathematik*	7
PGAS	partitioned global address space	8
WE	workflow engine	8

Index

Erklärung zu dieser Arbeit

Hiermit erkläre ich, die vorliegende Masterarbeit selbstständig erstellt zu haben. Weiterhin versichere ich, dass sämtliche von mir verwendeten Hilfsmittel und Quellen im Literaturverzeichnis aufgeführt sind.

Christian Reinbold, Hannover, den 11.01.2018