Evaluation and Optimization of Unate Covering Algorithms

Bernd Steinbach\(^1\) and Christian Posthoff\(^2\)

\(^1\) Freiberg University of Mining and Technology, Institute of Computer Science, D-09596 Freiberg, Germany
\(^2\) The University of The West Indies, St. Augustine Campus, Trinidad & Tobago

Abstract. The calculation of an exact minimal cover of a Boolean function is an \(NP\)-complete problem, which has an important application in circuit design. We could reduce the required time for the calculation by a factor of more than \(3.5 \times 10^7\) in [6], more than \(8 \times 10^8\) in [5], and even \(9 \times 10^9\) using a single CPU-core and finally \(1.2 \times 10^{11}\) using a GPU in [4]. In this paper we compare our so far best approach with a powerful algorithm for the same problem recently published by other authors [1]. The second aim of this paper is the optimization of our so far fastest algorithm which solves the Unate Covering Problem on a GPU.

1 Introduction

The Unate Covering Problem (UCP) must be solved to find minimal sets of prime conjunctions which completely cover the required Boolean function of a circuit. Consequently, the circuit space can be saved and the power consumption can be reduced.

We used in our previous papers [4–6] Algorithm 1 (ABS(DIST(\(P(p)\))) of [4] for simple direct comparisons. This algorithm is already \(10^4\) times faster than the a trivial UCP-algorithm. Here, we compare our so far fastest approach that utilizes a GPU with an new optimized algorithm from the literature. Despite the evaluated good performance of our algorithm, we explore three more approaches which even more reduce the run-time to 33 percent.

2 Unate Covering - the Problem

We consider a special SAT-problem without negated variables. Such functions are called Petrick functions \(P(p)\). The Petrick function defined by (1) depends on 8 variables and is given by 8 clauses:

\[
\begin{align*}
(p_4 \lor p_5 \lor p_6 \lor p_8) \land (p_2 \lor p_3 \lor p_4 \lor p_7 \lor p_8) \land (p_1 \lor p_3 \lor p_4 \lor p_7 \lor p_8) \\
(p_1 \lor p_4 \lor p_5 \lor p_7 \lor p_8) \land (p_1 \lor p_2 \lor p_4 \lor p_6) \land (p_4 \lor p_5 \lor p_6 \lor p_7 \lor p_8) \land \\
(p_1 \lor p_4 \lor p_5 \lor p_6 \lor p_7 \lor p_8) \land (p_4 \lor p_6 \lor p_7) = 1.
\end{align*}
\]  

(1)

The classical approach to solve the unate covering problem applies the distributive law [3] to the clauses, simplifies the created conjunctions using the idempotence law [3], and reduces the found disjunctive form using the absorption law [3].
3 Evaluation of UCP-Algorithms

Borowik and Luba published recently an improved algorithm [1] that solves the UCP using a tree based complement algorithm which was basically suggested in [2]. In order to exclude all side effects we used the original programs \texttt{exact\_reduct\_calculator.exe} and \texttt{obve\_g40.exe} on the same computer and the data mining benchmarks not prepared by ourselves.

Our program \texttt{obve\_g40.exe} runs for the data mining benchmarks of 9 to 22 variables which contain 81 to 1,900 clauses on the same computer 43 to 949,555 times faster.

4 Optimization of the Fastest GPU-Algorithm

We used one GPU NVIDIA® GeForce GTX 690 with Kepler architecture as hardware and the so far largest evaluated benchmark of [4] with 32 variables and 1024 clauses as basis for further improvements. Using the CUDA warp vote function \texttt{all()} and excluding each passive thread we could reduce the calculation time to 84 percent. Accepting as certain amount of passive threads but extends the number of thread within the blocks to an optimal number of warps, we even could reduce the calculation time to 33 percent.

5 Acknowledgment

We thank Grzegorz Borowik for providing us his executable program that solves the UCP using the algorithm presented in [1].

References