ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at SciVerse ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

The complexity of register allocation☆

Philipp Klaus Krause

Goethe-Universität, Institut für Informatik, Robert-Mayer-Straße 11-15, 60325 Frankfurt am Main, Germany

ARTICLE INFO

Article history:
Received 15 March 2012
Received in revised form 13 March 2013
Accepted 14 March 2013
Available online xxxx

Keywords:
Register allocation
Parametrized complexity
Structured program

ABSTRACT

In compilers, register allocation is one of the most important stages with respect to optimization for typical goals, such as code size, code speed, or energy efficiency. Graph theoretically, optimal register allocation is the problem of finding a maximum weight r-colorable induced subgraph in the conflict graph of a given program. The parameter r is the number of registers.

Large classes of programs are structured, i.e. their control-flow graphs have bounded tree-width (Thorup (1998) [17], Gustedt et al. (2002) [8] and Burgstaller et al. (2004) [3]). The decision problem of deciding if a conflict graph of a structured program is r-colorable is known to be fixed-parameter tractable (Bodlaender et al. (1998) [1]). Optimal register allocation for structured programs is known to be in XP (Krause (2013) [13]).

We complement these results by showing that optimal register allocation parametrized by r is W[SAT]-hard. This even holds for programs using only if/else and while as control structures; these programs form are subclass of the structured programs.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Register allocation is a compiler stage that tries to assign variables in a computer program to hardware registers in a processor. Variables that are alive at the same time (conflicting variables) cannot be assigned to the same register, since this would result in values that are still needed being overwritten. Variables that are not assigned to registers are stored in main memory instead, which typically is slower by several orders of magnitude, and takes more or longer instructions to access. Register allocation is one of the most important stages in a compiler with respect to optimization for typical goals, such as code size, code speed or energy efficiency.

Register allocation can be seen as coloring the conflict graph of the variables of the program, with colors being the available registers. For r registers, finding an r-colorable induced subgraph of maximum weight in the conflict graph is a simplification of the register allocation problem.

Large classes of programs are structured [17,8,3]. The problem of deciding if a conflict graph of a structured program is r-colorable is known to be fixed-parameter tractable [1]. Optimal register allocation for structured programs is known to be in XP [13].

We prove that finding an r-colorable induced subgraph of maximum weight in the conflict graph is W[SAT]-hard, even for a subclass of structured programs. This is a negative result complementing the earlier positive results.

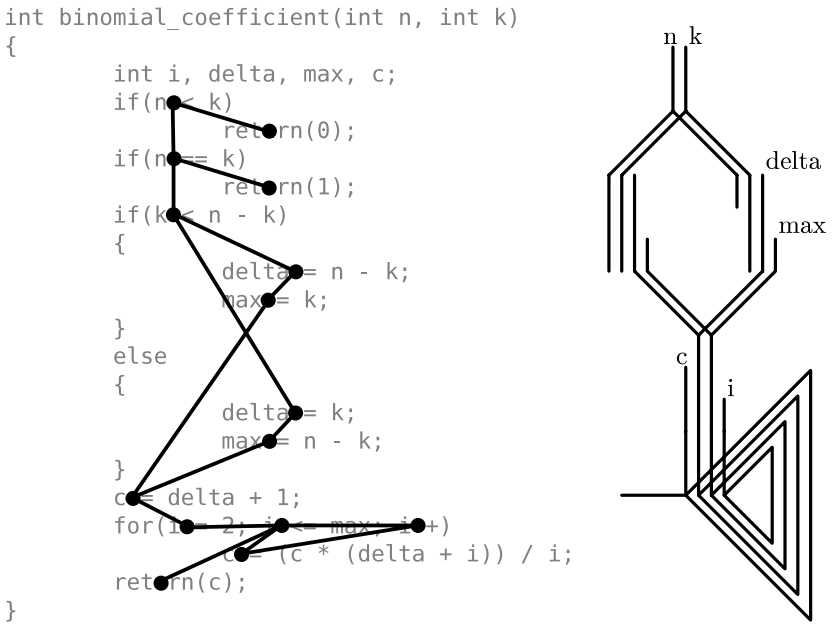
The following Section 2 introduces the basic concepts necessary for the discussion of the related work in Section 3 and our results in Section 4. Section 5 concludes and states some questions that are still open.

☆ This research was supported by DFG-Projekt GalA, grant number AD 411/1-1. It was partially conducted at NII, Tokyo, supported by a fellowship within the FIT-Programme of the German Academic Exchange Service (DAAD).

E-mail address: philipp@informatik.uni-frankfurt.de.

0166-218X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dam.2013.03.015

Please cite this article in press as: P.K. Krause, The complexity of register allocation, Discrete Applied Mathematics (2013), http://dx.doi.org/10.1016/j.dam.2013.03.015



(a) Code and CFG. (b) Variable live-ranges.

Fig. 1. Some program.

2. Preliminaries

Definition 1 (*Graph*)**.** A *graph* G is a pair of two sets, the *node set* and the *edge set*. For an *undirected graph* the edge set contains sets of two distinct nodes. For a *directed graph*, the edge set contains pairs of nodes.

Definition 2 (*Program*)**.** A *program* consists of a directed graph G, called the control-flow graph (CFG) of the program, a set of variables V and a weight function $c\colon V \to]0, \infty[$. Each node of G is marked by a subset of V. The set of nodes of G that are marked by a variable $v \in V$ are the *live-range* of v; v is said to be *alive* there. A live-range induces a connected subgraph of G.

This representation can be easily generated from other representations, such as pseudocode. The nodes of the CFG are the program's instructions; there is an edge from i to j, iff there is some execution of the program where instruction j is executed directly after instruction i. Typically there is a cost (code size, runtime, energy consumption) associated with not placing variables in registers. The cost depends on how often the variable is accessed in the program. This is represented in the weight function.

Fig. 1 shows code and CFG of a program and corresponding live-ranges.

Definition 3 (*Intersection Graph*)**.** Let S_I be a family of sets for some index set I. The *intersection graph* of S_I is the undirected graph that has a node for each $i \in I$, and two nodes are connected by an edge, if the intersection of the corresponding sets is nonempty:

$$\left(\{v_i | i \in I\}, \{\{v_i, v_j\} | S_i \cap S_j \neq \emptyset\}\right).$$

Definition 4 (*Conflict Graph*)**.** Let V be the set of variables of a program. The *conflict graph* of the program is the intersection graph of their live-ranges.

We use G for the control-flow graph and V for the variables throughout.

Definition 5 (*Series–Parallel Graph*)**.** A two-terminal graph (TTG) is a graph with two distinguished nodes, source s and sink t. The parallel composition of two TTGs can be obtained by taking their disjoint union, and then merging the two sources into the new source and merging the two sinks into the new sink. The series composition of two TTGs X and Y can be obtained by taking their disjoint union, and then merging the sink of X with the source of Y. The source for X and the sink of Y become source and sink of the new graph. The graphs that can be obtained from the TTGs that have a single edge connecting source and sink by doing parallel and series compositions, and forgetting about the distinction of source and sink are the *series–parallel* graphs.

Please cite this article in press as: P.K. Krause, The complexity of register allocation, Discrete Applied Mathematics (2013), http://dx.doi.org/10.1016/j.dam.2013.03.015

Download English Version:

https://daneshyari.com/en/article/6872239

Download Persian Version:

https://daneshyari.com/article/6872239

Daneshyari.com