Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 167-169

ISSN 2347 – 7393

Review article

CODE OPTIMIZATION AND GENERATION

Naeem Akhtar, Rahul, Naveen Malik, Pankaj Sharma, Hardeep Rohilla

Dronacharya College of Engineering, Khentawas,
Farukhnagar, Gurgaon, India

Abstract

CodeGeneration is the process of transforming code from one representation to another**..CodeOptimization** is the field where most compiler research is done today. The tasks of the front-end (scanning, parsing, semantic analysis) are well understood and unoptimizedcode generation is relatively straightforward. Optimization, on the other hand, still retains a sizable measure of mysticism. High-quality optimization is more of an art than a science. Compilers for mature languages aren't judged by how well they parse or analyze the code—you just expect it to do it right with a minimum of hassle—but instead by the quality of the object code they produce.

1. Introduction

In computer science, **code generation** is the process by which a compiler's code generator converts some intermediate representation of source code into a form (e.g., machine code) that can be readily executed by a machine.

Code Optimization is the process of transforming a piece of code to make more efficient (either in terms of time or space) without changing its output or side-effects. The only difference visible to the code'suser should be that it runs faster and/or consumes less memory. It is really a misnomer that the name implies you are finding an "optimal" solution— in truth, optimization aims to improve, not perfect, the result.

Many optimization problems are NP-complete and thus most optimization algorithms rely on heuristics and approximations. It may be possible to come up with a case where a particular algorithm fails to produce better code or perhaps even makes it worse. However, the algorithms tend to do rather well overall. It's worth reiterating here that efficient code starts with intelligent decisions by the programmer. No one expects a compiler to replace BubbleSort with Quicksort. If a programmer uses a lousy algorithm, no amount of optimization can make it snappy. In terms of big-O, a compiler can only make improvements to constant

For Correspondence:
malik2008@in.com
Received on: November 2013
Accepted after revision: December 2013

factors. But, all else being equal, you want an algorithm with low constant factors

In a compiler that uses an intermediate language, there may be two instruction selection stages — one to convert the parse tree into intermediate code, and a second phase much later to convert the intermediate code into instructions from the instruction set of the target machine. This second phase does not require a tree traversal; it can be done linearly, and typically involves a simple replacement of intermediate-language operations with their corresponding opcodes. However, if the compiler is actually a language translator (for example, one that converts Eiffel to C), then the second code-generation phase may involve building a tree from the linear intermediate code

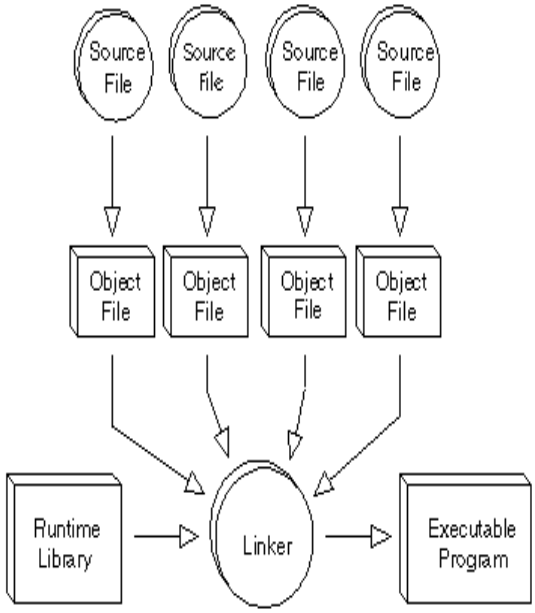
2. Object Code

Object code is the output of a compiler after it processes source code.Source code is the version of a computer program as it is originally written (i.e., typed into a computer) by a human in a programming language. A compiler is a specialized program that converts source code into object code.

The object code is usually a machine code, also called a machine language, which can be understood directly by a specific type of CPU (central processing unit), such as x86 (i.e., Intel-compatible) or PowerPC. However, some compilers are designed to convert source code into an assembly language or some other another programming language. An assembly language is a human-readable notation for the machine language that a specific type of CPU uses.code, or sometimes object module, is what a computer compiler produces.[1] In a general sense object code is a sequence of statements or instructions in a computer language,[2] usually a machine code language (i.e., 1's and 0's) or an intermediate language such as RTL.

Object files can in turn be linked to form executable file or library file. In order to be used, object code must either be placed in an executable file, a library file, or an object file.

Object code is a portion of machine code that hasn't yet been linked into a complete program. It's the machine code for one particular library or module that will make up the completed product. It may also contain



placeholders or offsets not found in the machine code of a completed program that the linker will use to connect everything together. Machine code is binary (1's and 0's) code that can be executed directly by the cpu. If you were to open a "machine code" file in a text editor you would see garbage, including unprintable characters. Object code is a variant of machine code, with a difference that the jumps are sort of parameterized such that a linker can fill them in. An assembler is used to convert assembly code into machine code (object code) A linker links several object (and library) files to generate an executable.

3. Machine Dependent Code

For some compiler, the intermediate code is a pseudo code of a virtual machine.

- Interpreter of the virtual machine is invoked to execute the intermediatecode.
- No machine-dependent code generation is needed.
- Usually with great overhead.
- Example:
  - . Pascal: P-code for the virtual P machine.
  - . JAVA: Byte code for the virtual JAVA machine

4. Machine code generation

- Input: intermediate code + symbol tables
- In our case, three-address code
- All variables have values that machines can directly manipulate
- Assume program is free of errors

 Type checking has taken place, type conversion done
 Output:
 Absolute/relocatable machine code or assembly code
 In our case, use assembly
 Architecture variations: RISC, CISC, stack-based
 Issues:
 Memory management, instruction selection and scheduling,register allocation and assignment

5. Approaches

 Top-down: count the number of references to each value
 the most heavily used values should reside in registers
 Weakness: dedicate a register to value for entire block
 Bottom-up: spill the value that is needed the latest
 For each variable use, compute the distance of its next use
 process each instruction in evaluation order; when running out of registers, spill the value whose next use is farthest in the future
 Produces excellent result in many cases
 Not optimal: not all spilling takes the same number of cycles
 Clean vs. dirty spill: has the variable been modified?
 Graph Coloring based allocation

6. References

- http://en.wikipedia.org/wiki/Code_generation_(compiler)
- http://c2.com/cgi/wiki?CodeGeneration
- http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/20-Optimization.pdf
- http://www.webopedia.com/TERM/O/object_code.html
- http://www.iis.sinica.edu.tw/~tshsu/compiler2007/slides/slide8.pdf
- http://www.cs.uccs.edu/~qyi/UTSA-classes/cs4713/slides/MCG.pdf