*CSU498 Project*

# Global Value Numbering

Kartik Singhal

(B090566CS)

Department of Computer Science and Engineering

National Institute of Technology Calicut

Wednesday November 7, 2012

# Outline

- Problem Definition
- Introduction
    - GVN
    - Some Known Algorithms
    - Motivation
- Work Done
    - Compiler Infrastructures
    - Structure of GCC
    - Intermediate Forms
    - Work Flow
    - A Naïve Implementation of Constant Propagation
- Conclusion & Future Work
- References

*Part 1*

# *Problem Definition*

# Problem Definition

To study global value numbering, a compiler optimization, in detail; to review and compare known algorithms; to implement one of the best among them; and in the process, improve upon the algorithm if possible.

# Problem Definition

To study global value numbering, a compiler optimization, in detail; to review and compare known algorithms; to implement one of the best among them; and in the process, improve upon the algorithm if possible.

# Problem Definition

To study global value numbering, a compiler optimization, in detail; to review and compare known algorithms; to implement one of the best among them; and in the process, improve upon the algorithm if possible.

# Problem Definition

To study global value numbering, a compiler optimization, in detail; to review and compare known algorithms; to implement one of the best among them; and in the process, improve upon the algorithm if possible.

*Part 2*

# *Introduction*

# GVN

Global value numbering (GVN) is a program analysis that categorizes expressions in the program that compute the same value. This information can be used to remove redundant computations.

# Some Known Algorithms

Kildall '73

Introduced a precise global analysis algorithm for program optimization by using the concept of an optimizing function for generalization.

AWZ '88

Introduced an efficient algorithm, uses a value graph to represent symbolic execution of program. Represents the values of variables after a join using a selection function $\phi$, as in SSA, treats them as uninterpreted, hence remains incomplete.

# Some Known Algorithms

RKS '99

Introduced polynomial time algorithm, extended the algorithm by AWZ. Employs a normalization process using some rewrite rules for terms involving $\phi$ functions, until congruence classes reach a fixed point. More equivalances, optimal for acyclic programs but remains incomplete.

Gargi '02

Proposed balanced algorithms, extends AWZ to perform forward propagation and reassociation and to consider back edges in SSA graph. Discovers more equivalences but is still incomplete.

Gulwani '04

Proposed a polynomial time algorithm which is optimal if only equalities of bounded size are considered.

# Motivation

- A compiler is a fairly large software program and forms an excellent software engineering case study.
- Optimizing compilers are hard to build.
- Study of compiler optimizations provides a good blend of theory (for generality and correctness) and practice (for validation and efficiency).
- Global Value Numbering, specifically, is an interesting global dataflow analysis for study.
- Opportunity to work on a popular open souce project's code base.

# Motivation

And . . .

Essential Abstractions in GCC '12 - A workshop on GCC Internals by GCC Resource Center, IIT Bombay.

*Part 3*

# *Work Done*

# Compiler Infrastructures

- SUIF - Stanford University Intermediate Format
- GCC - GNU Compiler Collection
- LLVM

We choose GCC as our implementation platform as it is a popular, professional, open-source compiler and could yield more realistic results.

# Structure of GCC

Conceptually three phases:

1. There is a separate front end for each supported language. A front end takes the source code, translates that source code into a semantically equivalent, language independent abstract syntax tree (AST). The syntax and semantics of this AST are defined by the GIMPLE language, the highest level language independent intermediate representation GCC has.

# Structure of GCC

Conceptually three phases:

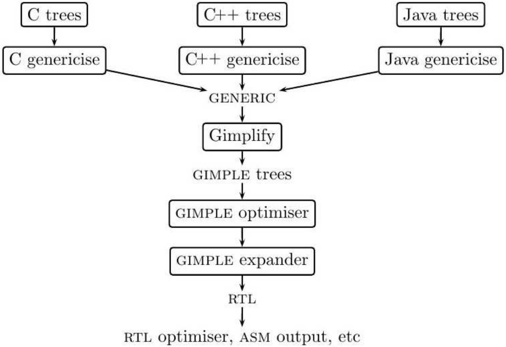
2. This AST is then run through a list of target independent code transformations that take care of constructing a control flow graph, and optimizing the AST for optimizing compilations, lowering to non-strict RTL (expand), and running RTL based optimizations for optimizing compilations. The non-strict RTL is handed over to more low-level passes.

# Structure of GCC

Conceptually three phases:

3. The low-level passes are the passes that are part of the code generation process. The first job of these passes is to turn the non-strict RTL representation into strict RTL. Other jobs of the strict RTL passes include scheduling, doing peephole optimizations, and emitting the assembly output.

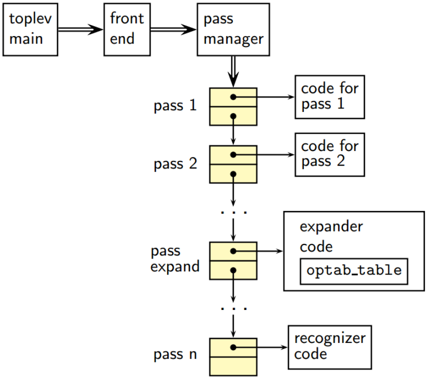
# Intermediate Forms



IR Framework in GCC

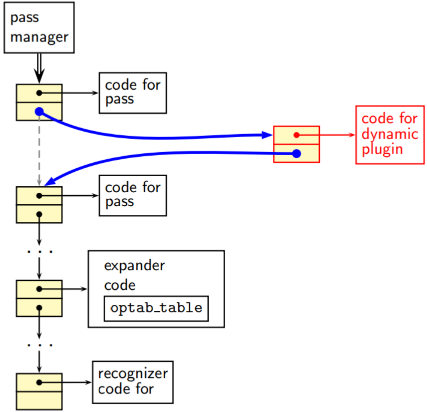
We work on GIMPLE which is a three-address intermediate form.

# Workflow



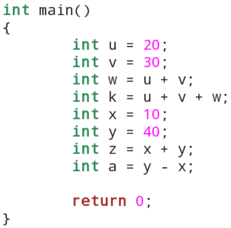
Plugin Structure in GCC

# Workflow



Dynamic Plugin Mechanism in GCC

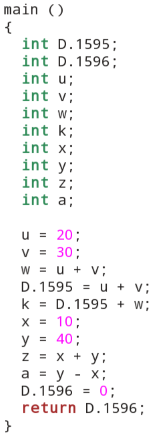
# A Naïve Implementation of Constant Propagation



```
int main()
{
        int u = 20;
        int v = 30;
        int w = u + v;
        int k = u + v + w;
        int x = 10;
        int y = 40;
        int z = x + y;
        int a = y - x;

        return 0;
}
```

Test program in C

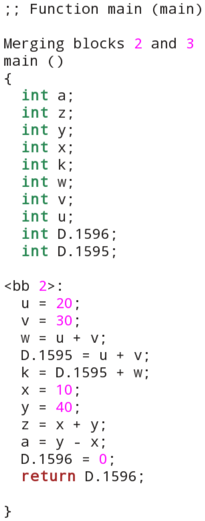
# A Naïve Implementation of Constant Propagation



```
main ()
{
  int D.1595;
  int D.1596;
  int u;
  int v;
  int w;
  int k;
  int x;
  int y;
  int z;
  int a;

  u = 20;
  v = 30;
  w = u + v;
  D.1595 = u + v;
  k = D.1595 + w;
  x = 10;
  y = 40;
  z = x + y;
  a = y - x;
  D.1596 = 0;
  return D.1596;
}
```

In GIMPLE IR form

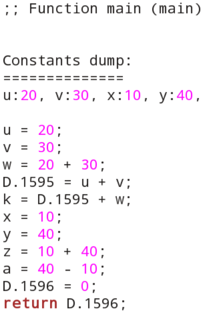
# A Naïve Implementation of Constant Propagation



```
;; Function main (main)

Merging blocks 2 and 3
main ()
{
  int a;
  int z;
  int y;
  int x;
  int k;
  int w;
  int v;
  int u;
  int D.1596;
  int D.1595;

<bb 2>:
  u = 20;
  v = 30;
  w = u + v;
  D.1595 = u + v;
  k = D.1595 + w;
  x = 10;
  y = 40;
  z = x + y;
  a = y - x;
  D.1596 = 0;
  return D.1596;

}
```

After CFG pass

# A Naïve Implementation of Constant Propagation



```
;; Function main (main)


Constants dump:
==============
u:20, v:30, x:10, y:40,

u = 20;
v = 30;
w = 20 + 30;
D.1595 = u + v;
k = D.1595 + w;
x = 10;
y = 40;
z = 10 + 40;
a = 40 - 10;
D.1596 = 0;
return D.1596;
```

Dump produced by our plugin

*Part 4*

# *Conclusion & Future Work*

# Conclusion & Future Work

Pros and cons of a number of known value numbering algorithms have been evaluated and familiarity with functionality of GCC as a compiler research infrastructure gained.

We plan to use GCC for implementation of a global value numbering algorithm, test and compare its performance with other known algorithms and look for any possible improvements in the algorithm during the remaining course of the project.

*Part 5*

# *References*

# References

- Gary A. Kildall. 1973. A unified approach to global program optimization. In *Proceedings of the 1st annual ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (POPL '73). ACM, 194-206. `http://doi.acm.org/10.1145/512927.512945`
- B. Alpern, M. N. Wegman, and F. K. Zadeck. 1988. Detecting equality of variables in programs. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (POPL '88). ACM, 1-11. `http://doi.acm.org/10.1145/73560.73561`
- Rüthing, O., Knoop, J., & Steffen, B. 1999. Detecting equalities of variables: Combining efficiency with precision. *Static Analysis*, 848-848.

# References

- Karthik Gargi. 2002. A sparse algorithm for predicated global value numbering. In *Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation* (PLDI '02). ACM, 45-56. http://doi.acm.org/10.1145/512529.512536
- Gulwani, S., & Necula, G. 2004. A polynomial-time algorithm for global value numbering. *Static Analysis*, 703-1020.
- VanDrunen, T. J. 2004. Partial redundancy elimination for global value numbering (Doctoral dissertation, Purdue University).
- Essential Abstractions in GCC '12 - A workshop on GCC Internals by GCC Resource Center, IIT Bombay. http://www.cse.iitb.ac.in/grc/gcc-workshop-12
- Structure of GCC - http://gcc.gnu.org/wiki/StructureOfGCC

# Thank You