WIKIPEDIA

Dead code elimination

In compiler theory, **dead code elimination** (also known as **DCE**, **dead code removal**, **dead code stripping**, or **dead code strip**) is a compiler optimization to remove code which does not affect the program results. Removing such code has several benefits: it shrinks program size, an important consideration in some contexts, and it allows the running program to avoid executing irrelevant operations, which reduces its running time. It can also enable further optimizations by simplifying program structure. *Dead code* includes code that can never be executed (*unreachable code*), and code that only affects *dead variables* (written to, but never read again), that is, irrelevant to the program.

Contents

Examples

Consider the following example written in C.

```c
int foo(void)
{
  int a = 24;
  int b = 25; /* Assignment to dead variable */
  int c;
  c = a * 4;
  return c;
  b = 24; /* Unreachable code */
  return 0;
}
```

Simple analysis of the uses of values would show that the value of `b` after the first assignment is not used inside `foo`. Furthermore, `b` is declared as a local variable inside `foo`, so its value cannot be used outside `foo`. Thus, the variable `b` is *dead* and an optimizer can reclaim its storage space and eliminate its initialization.

Furthermore, because the first return statement is executed unconditionally, no feasible execution path reaches the second assignment to `b`. Thus, the assignment is *unreachable* and can be removed. If the procedure had a more complex control flow, such as a label after the return statement and a `goto` elsewhere in the procedure, then a feasible execution path might exist to the assignment to `b`.

Also, even though some calculations are performed in the function, their values are not stored in locations accessible outside the scope of this function. Furthermore, given the function returns a static value (96), it may be simplified to the value it returns (this simplification is called constant folding).

Most advanced compilers have options to activate dead code elimination, sometimes at varying levels. A lower level might only remove instructions that cannot be executed. A higher level might also not reserve space for unused variables. Yet a higher level might determine instructions or functions that serve no purpose and eliminate them.

A common use of dead code elimination is as an alternative to optional code inclusion via a preprocessor. Consider the following code.

```
int main(void) {
  int a = 5;
  int b = 6;
  int c;
  c = a * (b / 2);
  if (0) {   /* DEBUG */
    printf("%d\n", c);
  }
  return c;
}
```

Because the expression 0 will always evaluate to false, the code inside the if statement can never be executed, and dead code elimination would remove it entirely from the optimized program. This technique is common in debugging to optionally activate blocks of code; using an optimizer with dead code elimination eliminates the need for using a preprocessor to perform the same task.

In practice, much of the dead code that an optimizer finds is created by other transformations in the optimizer. For example, the classic techniques for operator strength reduction insert new computations into the code and render the older, more expensive computations dead.[1] Subsequent dead code elimination removes those calculations and completes the effect (without complicating the strength-reduction algorithm).

Historically, dead code elimination was performed using information derived from data-flow analysis.[2] An algorithm based on static single assignment form (SSA) appears in the original journal article on *SSA* form by Ron Cytron et al.[3] Robert Shillingsburg (aka Shillner) improved on the algorithm and developed a companion algorithm for removing useless control-flow operations.[4]

Dynamic dead code elimination

Dead code is normally considered dead *unconditionally*. Therefore, it is reasonable attempting to remove dead code through dead code elimination at compile time.

However, in practice it is also common for code sections to represent dead or unreachable code only *under certain conditions*, which may not be known at the time of compilation or assembly. Such conditions may be imposed by different runtime environments (for example different versions of an operating system, or different sets and combinations of drivers or services loaded in a particular target environment), which may require different sets of special cases in the code, but at the same time become conditionally dead code for the other cases.[5][6] Also, the software (for example, a driver or resident service) may be configurable to include or exclude certain features depending on user preferences, rendering unused code portions useless in a particular scenario.[5][6] While modular software may be developed to dynamically load libraries on demand only, in most cases, it is not possible to load only the relevant routines from a particular library, and even if this would be supported, a routine may still include code sections which can be considered dead code in a given scenario, but could not be ruled out at compile time, already.

The techniques used to dynamically detect demand, identify and resolve dependencies, remove such conditionally dead code, and to recombine the remaining code at load or runtime are called **dynamic dead code elimination**[5][6][7][8][9][10][11][12][13][14][15][16][17] or **dynamic dead instruction elimination**.[18]

Most programming languages, compilers and operating systems offer no or little more support than dynamic loading of libraries and late linking, therefore software utilizing dynamic dead code elimination is very rare in conjunction with languages compiled ahead-of-time or written in assembly language.[7][11][8] However, language implementations doing just-in-time compilation may dynamically optimize for dead code elimination.[17][19][20]

Although with a rather different focus, similar approaches are sometimes also utilized for dynamic software updating and hot patching.

See also

- Redundant code
- Simplification (symbolic computation)
- Partial redundancy elimination
- Conjunction elimination
- Mathematical elimination
- Dynamic software updating
- Dynamic coupling (computing)
- Software cruft
- Tree shaking
- Post-pass optimization
- Profile-guided optimization
- Superoptimizer
- Compacting garbage collection
- Self-replication

References

1. Allen, Frances; Cocke, John; Kennedy, Ken (June 1981). "Reduction of Operator Strength". In Jones, Neil D.; Muchnick, Steven Stanley (eds.). *Program Flow Analysis: Theory & Application*. Prentice-Hall. ISBN 0-13729681-9.
2. Kennedy, Ken (June 1981). "A Survey of Data-flow Analysis Techniques". In Jones, Neil D.; Muchnick, Steven Stanley (eds.). *Program Flow Analysis: Theory & Application*. Prentice-Hall. ISBN 0-13729681-9.
3. Cytron, Ron K.; Ferrante, Jeanne; Rosen, Barry K.; Zadeck, F. Kenneth (1991). *Efficiently Computing Static Single Assignment Form and the Program Dependence Graph*. ACM TOPLAS 13(4).
4. Cooper, Keith D.; Torczon, Linda (2003) [2002-01-01]. *Engineering a Compiler*. Morgan Kaufmann. pp. 498ff. ISBN 978-1-55860698-2.

5. Paul, Matthias R. (2002-04-03) [2001-06-18]. "[fd-dev] Ctrl+Alt+Del" (https://marc.info/?l=freedos-dev&m=101783474625117). *freedos-dev*. Archived (https://archive.today/20170909084942/https://marc.info/?l=freedos-dev&m=101783474625117) from the original on 2017-09-09. Retrieved 2017-09-09. "[…] any of the […] options can be "permanently" excluded at installation time (will also save the memory for the corresponding code excerpts due to our Dynamic Dead Code Elimination), or it can be disabled or enabled at any later time via API functions in case someone wants to keep a user from being able to reboot the machine. […] we are considering to add more synchronous cache flush calls […] Due to our Dynamic Dead Code Elimination method this would not cause any kind of bloating when not needed in a particular target configuration as a particular cache flush call would be included in FreeKEYB's runtime image only if the corresponding disk cache is loaded as well or FreeKEYB was told by command line switches to load the corresponding support."
6. Paul, Matthias R. (2002-04-06). "[fd-dev] Ctrl+Alt+Del" (https://marc.info/?l=freedos-dev&m=101807225917568&w=2). *freedos-dev*. Archived (https://archive.today/20190427131940/https://marc.info/?l=freedos-dev&m=101807225917568&w=2) from the original on 2019-04-27. Retrieved 2019-04-27. "[…] FreeKEYB builds the driver's runtime image at initialization time depending on the type of machine it is being loaded on, the type of keyboard, layout, country and code page used, the type of mouse and video adapter(s) installed, the other drivers loaded on that system, the operating system and the load and relocation method(s) used, the individual features included, and the configuration options specified in the command line. Due to the large number of command line switches and options supported […] (around fifty switches […] with multiple possible settings) there is a high number of feature combinations with uncountable dependencies […] resulting in […] endless number of […] different target images. FreeKEYB's Dynamic Dead Code Elimination technique manages to resolve […] these […] dependencies and […] remove dead code and data […] is not restricted to […] include or exclude a somewhat limited number of modules or whole sub-routines and fix up some dispatch tables as in classical TSR programming, but […] works […] at […] byte level […] able to remove […] individual instructions in the middle of larger routines […] distributed all over the code to handle a particular case or support a specific feature […] special tools are used to analyze the code […] and create […] fixup tables […] automated […] using conditional defines […] to declare the various cases […] not only optional at assembly time but at initialization time […] without the […] overhead of having at least some amount of dead code left in the runtime image […] to keep track of all the dependencies between […] these conditionals, dynamically build and relocate the runtime image, fix up all the references between these small, changing, and moving binary parts […] still allowing to use the tiny .COM/.SYS style […] model […] is done at initialization time […] API to import and export object structures between FreeKEYB and the calling application […] to transparently resize and move them internally […] at runtime […]"
7. Paul, Matthias R.; Frinke, Axel C. (1997-10-13) [first published 1991], *FreeKEYB - Enhanced DOS keyboard and console driver* (User Manual) (v6.5 ed.) [1] (https://web.archive.org/web/20190309194320/http://sta.c64.org/dosprg/fk657p1.zip) (NB. FreeKEYB is a Unicode-based dynamically configurable successor of K3PLUS supporting most keyboard layouts, code pages, and country codes. Utilizing an off-the-shelf macro assembler as well as a framework of automatic pre- and post-processing analysis tools to generate dependency and code morphing meta data to be embedded into the executable file alongside the binary code and a self-discarding, relaxing and relocating loader, the driver implements byte-level granular dynamic dead code elimination and relocation techniques at load-time as well as self-modifying code and reconfigurability at run-time to minimize its memory footprint downto close the canonical form depending on the underlying hardware, operating system, and driver configuration as well as the selected feature set and locale (about sixty configuration switches with hundreds of options for an almost unlimited number of possible combinations). This complexity and the dynamics are hidden from users, who deal with a single executable file just like they would do with a conventional driver. K3PLUS was an extended keyboard driver for DOS widely distributed in Germany at its time, with adaptations to a handful of other European languages available. It supported a sub-set of features already, but did not implement dynamic dead code elimination.)

8. Paul, Matthias R. (2001-04-10). "[ANN] FreeDOS beta 6 released" (https://groups.google.com/d/msg/de.comp.os.msdos/qCZs8p6MyPQ/Pksl0Pv6qM8J) (in German). Newsgroup: de.comp.os.msdos (news:de.comp.os.msdos). Archived (https://archive.today/20170909084250/https://groups.google.com/forum/%23!msg/de.comp.os.msdos/qCZs8p6MyPQ/Pksl0Pv6qM8J) from the original on 2017-09-09. Retrieved 2017-07-02. "[…] brandneue[s] Feature, der dynamischen Dead-Code-Elimination, die die jeweils notwendigen Bestandteile des Treibers erst zum Installationszeitpunkt zusammenbastelt und reloziert, so daß keine ungenutzten Code- oder Datenbereiche mehr resident bleiben (z.B. wenn jemand ein bestimmtes FreeKEYB-Feature nicht benötigt). […]" (NB. This represents the first known implementation of byte-level granular *dynamic dead code elimination* for software assembled or compiled ahead-of-time.)
9. Paul, Matthias R. (2001-08-21). "[fd-dev] Changing codepages in FreeDOS" (https://marc.info/?l=freedos-dev&m=99840256128898&w=2). *freedos-dev*. Archived (https://web.archive.org/web/20190419120006/https://marc.info/?l=freedos-dev&m=99840256128898&w=2) from the original on 2019-04-19. Retrieved 2019-04-20. "[…] a […] unique feature […] we call dynamic dead code elimination, so you can at installation time […] specify which components of the driver you want and which you don't. This goes to an extent of dynamic loadable modularization and late linkage I have not seen under DOS so far. If you do not like the screen saver, macros, the calculator, or mouse support, or <almost anything else>, you can specify this at the command line, and FreeKEYB, while taking all the dependencies between the routines into account, will completely remove all the code fragments, which deal with that feature and are not necessary to provide the requested functionality, before the driver relocates the image into the target location and makes itself resident. Removing some smaller features just saves a couple of bytes but excluding more complex components can save you half a Kb and more. You can also specify the size of the data areas […]"
10. Paul, Matthias R. (2001-12-30). "KEYBOARD.SYS internal structure" (https://groups.google.com/d/msg/comp.os.msdos.programmer/l_IuSHsBDWQ/887rJF9IYmMJ). Newsgroup: comp.os.msdos.programmer (news:comp.os.msdos.programmer). Archived (https://archive.today/20170909082257/https://groups.google.com/forum/%23!msg/comp.os.msdos.programmer/l_IuSHsBDWQ/887rJF9IYmMJ) from the original on 2017-09-09. Retrieved 2017-07-03. "[…] the loader will dynamically optimize the resident data areas and code sections at byte level to remove any redundancy from the driver depending on the given hardware/software/driver configuration and locale. […]"
11. Paul, Matthias R.; Frinke, Axel C. (2006-01-16), *FreeKEYB - Advanced international DOS keyboard and console driver* (User Manual) (v7 preliminary ed.)
12. Paul, Matthias R. (2002-02-02). "Treiber dynamisch nachladen (Intra-Segment-Offset-Relokation zum Laden von TSRs in die HMA)" (https://groups.google.com/d/msg/de.comp.os.msdos/tdvpBoMVN6A/a_zqDxGk22IJ) [Loading drivers dynamically (Intra-segment offset relocation to load TSRs into the HMA)] (in German). Newsgroup: de.comp.os.msdos (news:de.comp.os.msdos). Archived (https://archive.today/20170909085006/https://groups.google.com/forum/%23!msg/de.comp.os.msdos/tdvpBoMVN6A/a_zqDxGk22IJ) from the original on 2017-09-09. Retrieved 2017-07-02.
13. Paul, Matthias R. (2002-02-24). "GEOS/NDO info for RBIL62?" (https://groups.google.com/d/msg/comp.os.geos.programmer/8NNPJ4VU23A/cucVV95kf9oJ). Newsgroup: comp.os.geos.programmer (news:comp.os.geos.programmer). Archived (https://archive.today/20190420111421/https://groups.google.com/forum/%23!msg/comp.os.geos.programmer/8NNPJ4VU23A/cucVV95kf9oJ) from the original on 2019-04-20. Retrieved 2019-04-20. "[…] Since FreeKEYB uses our dynamic dead-code-elimination to optimize its memory image for the target environment at load time, I would certainly like to add special support into FreeKEYB for GEOS which could be controlled by a command line option, so the extra code is only loaded when GEOS is used as well. […]"

14. Paul, Matthias R. (2002-03-15). "AltGr keyboard layer under GEOS?" (https://groups.google.com/d/msg/comp.os.geos.misc/THFVa3zvlzI/NqN2b1aBZfsJ). Newsgroup: comp.os.geos.misc (news:comp.os.geos.misc). Archived (https://archive.today/20190420111906/https://groups.google.com/forum/%23!msg/comp.os.geos.misc/THFVa3zvlzI/NqN2b1aBZfsJ) from the original on 2019-04-20. Retrieved 2019-04-20. "[…] I would be willing to add special hooks into FreeKEYB, our much advanced DOS keyboard driver, would it prove to improve the usability under GEOS […] Due to our sophisticated new Dynamic Dead Code Elimination technology which removes at byte level any code snippets unused in a particular given driver, user, or system configuration and hardware environment when the driver's installer builds and relocates the load image of itself, this would have no memory impact for non-GEOS users at all, so there's not much to worry about (memory footprint etc.) as in traditionally coded DOS drivers. […]"
15. Thammanur, Sathyanarayan (2001-01-31). *A Just in Time Register Allocation and Code Optimization Framework for Embedded Systems* (http://rave.ohiolink.edu/etdc/view?acc_num=ucin982089462) (MS thesis). University of Cincinnati, Engineering: Computer Engineering. ucin982089462. [2] (https://etd.ohiolink.edu/!etd.send_file?accession=ucin982089462&disposition=inline)[3] (https://etd.ohiolink.edu/!etd.send_file?accession=ucin982089462&disposition=attachment)
16. Glew, Andy (2011-03-02). "Dynamic dead code elimination and hardware futures" (http://semipublic.comp-arch.net/wiki/Dynamic_dead_code_elimination_and_hardware_futures). [4] (https://groups.google.com/d/msg/comp.arch/uJsmleaAr6o/pmbytnt02XcJ) [5] (http://blog.andy.glew.ca/2011/03/)
17. Conway, Andrew (1995-12-04). "Cyclic data structures" (https://groups.google.com/d/msg/comp.lang.functional/zFsRAGroeVM/2L5TIgcG-1IJ). Newsgroup: comp.lang.functional (news:comp.lang.functional). Archived (https://archive.today/20170909085605/https://groups.google.com/forum/%23!msg/comp.lang.functional/zFsRAGroeVM/2L5TIgcG-1IJ) from the original on 2017-09-09. Retrieved 2017-07-03. "[…] Lazy evaluation is basically dynamic dead code elimination. […]" (NB. Possibly the first public use of the term *dynamic dead code elimination*, though only conceptually and with a focus on lazy evaluation in functional languages.)
18. Butts, J. Adam; Sohi, Guri (October 2002). "Dynamic Dead-Instruction Detection and Elimination" (ftp://ftp.cs.wisc.edu/sohi/papers/2002/deadinstructions.asplos.pdf) (PDF). San Jose, CA, USA: Computer Science Department, University of Wisconsin-Madison. ASPLOS X ACM 1-58113-574-2/02/0010. Archived (https://archive.today/20190420144100/http://ftp.cs.wisc.edu/sohi/papers/2002/deadinstructions.asplos.pdf) (PDF) from the original on 2019-04-20. Retrieved 2017-06-23.
19. Johng, Yessong; Danielsson, Per; Ehnsiö, Per; Hermansson, Mats; Jolanki, Mika; Moore, Scott; Strander, Lars; Wettergren, Lars (2002-11-08). "Chapter 5. Java overview and iSeries implementation - 5.1.1. Miscellaneous components". *Intentia Movex Java on the IBM iSeries Server - An Implementation Guide - Overview of Movex Java on the iSeries server - Movex Java on iSeries installation and configuration - Operational tips and techniques* (https://www.redbooks.ibm.com/redbooks/pdfs/sg246545.pdf) (PDF). Red Books. IBM Corp. p. 41. ISBN 0-73842461-7. SG24-6545-00. Archived (https://web.archive.org/web/20131008194558/http://www.redbooks.ibm.com/redbooks/pdfs/sg246545.pdf) (PDF) from the original on 2013-10-08. Retrieved 2019-04-20. [6] (https://www.redbooks.ibm.com/abstracts/sg246545.html)
20. Polito, Guillermo (2015). "Virtualization Support for Application Runtime Specialization and Extension - Programming Languages" (https://hal.inria.fr/tel-01251173/file/Poli15Thesis.pdf) (PDF). Universite des Sciences et Technologies de Lille. pp. 111–124. HAL Id: tel-01251173. Archived (https://web.archive.org/web/20170623021500/https://hal.inria.fr/tel-01251173/file/Poli15Thesis.pdf) (PDF) from the original on 2017-06-23. Retrieved 2017-06-23.

Further reading

- Bodík, Rastislav; Gupta, Rajiv (June 1997). "Partial dead code elimination using slicing transformations". *Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation (PLDI '97)*: 682–694.
- Aho, Alfred Vaino; Sethi, Ravi; Ullman, Jeffrey David (1986). *Compilers - Principles, Techniques and Tools* (https://archive.org/details/compilersprincip00ahoa). Addison Wesley Publishing Company. ISBN 0-201-10194-7.
- Muchnick, Steven Stanley (1997). *Advanced Compiler Design and Implementation* (https://archive.org/details/advancedcompiler00much). Morgan Kaufmann Publishers. ISBN 1-55860-320-4.
- Grune, Dick; Bal, Henri Elle; Jacobs, Ceriel J. H.; Langendoen, Koen G. (2000). *Modern Compiler Design*. John Wiley & Sons, Inc. ISBN 0-471-97697-0.
- Kennedy, Ken; Allen, Randy (2002). "Chapter 4.4. Data Flow Analysis - Chapter 4.4.2. Dead Code Elimination". *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach* (https://archive.org/details/optimizingcompil00alle_837) (2011 digital print of 1st ed.). Academic Press / Morgan Kaufmann Publishers / Elsevier. pp. 137 (https://archive.org/details/optimizingcompil00alle_837/page/n134), 145–147, 167. ISBN 978-1-55860-286-1. LCCN 2001092381 (https://lccn.loc.gov/2001092381).

External links

- How to trick C/C++ compilers into generating terrible code? (https://archive.today/20130112201318/http://www.futurechips.org/tips-for-power-coders/how-to-trick-cc-compilers-into-generating-terrible-code.html)