Common subexpression elimination

From Wikipedia, the free encyclopedia

In compiler theory, common subexpression elimination (CSE) is a compiler optimization that searches for instances of identical expressions (i.e., they all evaluate to the same value), and analyzes whether it is worthwhile replacing them with a single variable holding the computed value.

Contents

Example

In the following code:

```
a = b * c + g;
d = b * c * e;
```

it may be worth transforming the code to:

```
tmp = b * c;
a = tmp + g;
d = tmp * e;
```

if the time cost (savings) of storing and retrieving "tmp" outweighs the cost of calculating "b * c" an extra time.

Principle

The possibility to perform CSE is based on available expression analysis (a data flow analysis). An expression `b*c` is available at a point *p* in a program if:

- every path from the initial node to p evaluates `b*c` before reaching *p*,
- and there are no assignments to `b` or `c` after the evaluation but before *p*.

The cost/benefit analysis performed by an optimizer will calculate whether the cost of the store to `tmp` is less than the cost of the multiplication; in practice other factors such as which values are held in which registers are also significant.

Compiler writers distinguish two kinds of CSE:

- local common subexpression elimination works within a single basic block
- global common subexpression elimination works on an entire procedure,

Both kinds rely on data flow analysis of which expressions are available at which points in a program.

Benefits

The benefits of performing CSE are great enough that it is a commonly used optimization.

In simple cases like in the example above, programmers may manually eliminate the duplicate expressions while writing the code. The greatest source of CSEs are intermediate code sequences generated by the compiler, such as for array indexing calculations, where it is not possible for the developer to manually intervene. In some cases language features may create many duplicate expressions. For instance, C macros, where macro expansions may result in common subexpressions not apparent in the original source code.

Compilers need to be judicious about the number of temporaries created to hold values. An excessive number of temporary values creates register pressure possibly resulting in spilling registers to memory, which may take longer than simply recomputing an arithmetic result when it is needed.

See also

- Global value numbering
- Loop-invariant code motion

- Compiler optimization

References

- Steven S. Muchnick, *Advanced Compiler Design and Implementation* (Morgan Kaufmann, 1997) pp. 378–396
- John Cocke. "Global Common Subexpression Elimination." (http://dl.acm.org/citation.cfm?id=390013.808480) *Proceedings of a Symposium on Compiler Construction*, *ACM SIGPLAN Notices* 5(7), July 1970, pages 850-856.
- Briggs, Preston, Cooper, Keith D., and Simpson, L. Taylor. "Value Numbering." *Software-Practice and Experience*, 27(6), June 1997, pages 701-724.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Common_subexpression_elimination&oldid=769415201"

Categories: Compiler optimizations

- This page was last edited on 9 March 2017, at 11:26.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.