Value-numbering on Regions

...and Beyond

Richard Biener
SUSE Labs
rguenther@suse.de

WANTED: fast cleanup after code-generation

Existing state:

- No cleanup (most passes).
- Use hash-tables or caches to do CSE (vectorizer).
- Own mini-pass (unroller).
- Schedule passes accordingly.

Problem with this:

- Imperfect or no cleanup confuses followup passes.
- Extra passes run unconditionally and cost too much.
- Maintainance issue with duplicate code to avoid leaking cleanup opportunities.

SOLUTION: some generic facility

What are the options?

- Try to make an existing cleanup work as utility (on smaller regions):
  - Successfully prototyped for single BBs piggy-backing on DOM.
  - Existing mini-passes (propagate_constants_for_unrolling)
- Implement something new!

Here comes RPO value-numbering

Turn existing SCC based value-numbering (FRE / PRE) into something that can work on regions. Solve issues we have with SCC VN at the same time:

- Expression simplification works on expressions of values; forces us to stow away SSA info to avoid miscompiles. But that's not enough as recent PRs show.
- SCC VN is an iteration scheme that handles CFG cycles not optimistically (but SSA cycles). We've retrofitted some capability on top of it but that's imperfect.
- It currently doesn't have a non-iterating mode which would be nice for -O1 or region processing.

At the same time our SCC VN has got loads of capabilities we want to preserve.

→ Change only the iteration scheme and API.

RPO VN API

unsigned do_rpo_vn (function *fn, edge entry, bitmap exits);

Value-number and eliminate the single entry, multiple exit (SEME) region with ENTRY edge and EXITS exit basic-blocks. Returns TODO_cleanup_cfg if the CFG needs to be cleaned up to make the IL consistent again.

- O(region size)* complexity in time and space
- Performs non-iterating value-numbering and elimination in a single pass over the IL
- Removes code that becomes dead as the result (but not basic-blocks or edges)

Current users:

- Loop unrolling and peeling on GIMPLE has its mini-pass replaced by this.

Future possible users:

- Vectorizer
- IVOPTs
- Jump threading
- ...

RPO VN use by FRE and PRE

- FRE at -O1 uses the not iterating mode
- FRE at -O2 and PRE use the iterating mode and use a separate delayed elimination path inherited from the old implementation

RPO VN features

RPO VN can handle CFG cycles optimistically which means it does not iterate when backedges can be computed as not executing.

RPO VN can disentangle irreducible regions in less passes.

RPO VN does simplification of expressions built from SSA names that are available leaders for their values at certain program points.

When iterating RPO VN is usually slower than SCC VN:

- CFG cycles are easily more complex than SSA cycles
- --param rpo-vn-max-loop-depth makes us handle regions of the CFG pessimistically, requiring no iteration
- RPO VN tracks availability for expression simplification

TODO

- **WIP patch for separating iteration and non-iteration**
- **Experiment with adding a VRP lattice**
- **Use from more places**
- **Replace DOM value-numbering**
- **Improve compile-time – 95% of the time spent in alias-walks**
  - DOM does better here by constraining what kind of memory operations it can CSE
  - Alias-walks could use edge executability for pruning the walks (currently done in a limited form by value-numbering of virtual operands)
  - Something as simple as not doing lookup again for stmts whose DEFs were already value-numbered as VARYING in the previous iteration (we still need to re-populate the hashes)

Q&A


SUSE
We adapt. You succeed.

Unpublished Work of SUSE LLC. All Rights Reserved.

This work is an unpublished work and contains confidential, proprietary and trade secret information of SUSE LLC. Access to this work is restricted to SUSE employees who have a need to know to perform tasks within the scope of their assignments. No part of this work may be practiced, performed, copied, distributed, revised, modified, translated, abridged, condensed, expanded, collected, or adapted without the prior written consent of SUSE.
Any use or exploitation of this work without authorization could subject the perpetrator to criminal and civil liability.

General Disclaimer

This document is not to be construed as a promise by any participating company to develop, deliver, or market a product. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. SUSE makes no representations or warranties with respect to the contents of this document, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. The development, release, and timing of features or functionality described for SUSE products remains at the sole discretion of SUSE. Further, SUSE reserves the right to revise this document and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes. All SUSE marks referenced in this presentation are trademarks or registered trademarks of Novell, Inc. in the United States and other countries. All third-party trademarks are the property of their respective owners.