MIR Support in llvm-reduce

Matt Arsenault

AMD
together we advance_

Motivation

- Deluge of assertions and verifier errors in register allocation
- Very sensitive to any minor code changes
- IR reduction alone is inadequate
  - Often cannot get below thousands of IR instructions
- These kinds of failures bit rot incredibly quickly
  - Important to get a minimal MIR reproducer quickly
- Fractal explosion of other failures found during reduction process

Usage

```
$ llvm-reduce -mtriple=amdgcn-amd-amdhsa -mcpu=gfx900 --test=./run_llc.sh
failure.mir
```

```
#!/bin/bash
! llc -mtriple=amdgcn-amd-amdhsa -mcpu=gfx900 -start-before=simple-register-
coalescing -stop-after=greedy,1 -verify-machineinstrs $@
```

```
#!/bin/bash
llc -mtriple=amdgcn-amd-amdhsa -mcpu=gfx900 -start-before=machine-scheduler -
stop-after=greedy,1 -verify-regalloc $@ 2>&1 | grep -m1 “Cannot decrease
cascade number, illegal eviction”
```

Reduction Implementation

```
static void extractInstrFromModule(Oracle &O, ReducerWorkItem &WorkItem) {
  for (const Function &F : WorkItem.getModule()) {
    if (MachineFunction *MF = WorkItem.MMI->getMachineFunction(F))
      extractInstrFromFunction(O, *MF);
  }
}
```

MIR vs. IR

- More difficult to produce plausibly valid MIR

- Really 3 IRs in one
  - Generic MIR
  - Selected SSA
  - Post-SSA

MIR vs. IR

- Virtual registers are not a direct replacement for Values
  - Cannot simply replace deleted values with undef/poison
  - Need to find a valid place to place IMPLICIT_DEF
- Need to consider register liveness
- Different control flow graph (CFG) representation
  - IR BasicBlocks implicitly track CFG through block references in terminator instructions
  - MachineBasicBlocks directly track successors and predecessors
  - Fall-through blocks
  - Terminators may not be analyzable
  - No undef blocks
    - Use dummy empty block inserted at the end of the function
- Additional properties not represented in the IR

Reduction Passes

- Instructions
- Reduce Uses / Defs
  - Instructions can have multiple results
  - Insert replacement IMPLICIT_DEFs
  - Delete implicit operands post SSA
- Reduce IR References - Eliminate IR references
  - MachineMemOperands
  - FrameIndexes derived from alloca
  - IR BasicBlock names
- Register hints
- Register masks
- Instruction flags

```
---
name: func
tracksRegLiveness: true
fixedStack:
  - { id: 0, offset: 16, size: 8, alignment: 8 }
stack:
  - { id: 0, size: 32, alignment: 8, name: alloca }
registers:
  - { id: 0, class: vreg_64, preferred-register: ‘$vgpr0_vgpr1' }
  - { id: 1, class: _, preferred-register: '' }
  - { id: 2, class: vreg_64, preferred-register: '%1' }
body:             |
  bb.0.entry:
  livens: $vgpr0_vgpr1
    S_WAITCNT 0
    %0:vreg_64(p1) = COPY $vgpr0_vgpr1
    %1:_(<2 x s32>) = G_LOAD %0 :: (load (<2 x s32>) from %ir.argptr0, align 32, addrspace 1)
    %2:vreg_64(<2 x s32>) = COPY %1

  bb.1.block.name.0:
    %3:_(<2 x s32>) = G_LOAD %0 :: (load (<2 x s32>) from %ir.argptr1, addrspace 3)
    %4:_(<2 x s32>) = G_LOAD %0 :: (load (<2 x s32>) from %ir.argptr1 + 8, addrspace 3)
    %5:_(<2 x s32>) = G_LOAD %0 :: (load (<2 x s32>) from %ir.argptr1 + 12, addrspace 3)
    %6:_(<2 x s32>) = G_ADD %2, %3
    %7:_(<2 x s32>) = nuw nsw G_ADD %6, %4
    %8:_(s32), %9:_(s1) = G_UADDO %6, %7
    %10:_(p5) = G_IMPLICIT_DEF
    %11:_(p5) = G_FRAME_INDEX %stack.0
    G_STORE %7, %10 :: (store (<2 x s32>) into %fixed-stack.0, addrspace 5)
    G_STORE %7, %11 :: (store (<2 x s32>) into %stack.0.alloca, addrspace 5)
    $sgpr30_sgpr31 = SI_CALL %13:sreg_64_xexec, 0, CustomRegMask($vgpr8_vgpr9,
$vgpr9_vgpr10_vgpr11,$vcc_lo,$agpr8,$sgpr99,$vgpr23,$vgpr48_vgpr49_vgpr50,$vgpr49_vgpr50_vgpr51,
$vgpr52_vgpr53_vgpr54,$vcc_hi,$sgpr0_sgpr1_sgpr2_sgpr3,$sgpr4_sgpr5_sgpr6_sgpr7), implicit %9
  bb.3.exit:
    %12:_(p5) = G_IMPLICIT_DEF
    %13:_(s32) = G_IMPLICIT_DEF
    G_STORE %13, %12 :: (store (s32) into %ir.keep.store, addrspace 5)
    S_ENDPGM 0, implicit %7
...
```

Target Support

- MachineFunctionInfo::clone
  - Most implementations trivial
- Register and frame index values remain unchanged
- Needs to remap any pointer values
  - MachineBasicBlocks

```
MachineFunctionInfo *
SIMachineFunctionInfo::clone(
  BumpPtrAllocator &Allocator,
  MachineFunction &DestMF,
  const DenseMap<MachineBasicBlock *,
MachineBasicBlock *> &Src2DstMBB) const {
    return
DestMF.cloneInfo<SIMachineFunctionInfo>(*this);
}
```

Machine Verifier Issues

- Verifier optionally checks LiveIntervals
  - Uses separate, buggier liveness checks without it
  - Fails to catch missing defs in the entry block
  - Fails to catch subregister issues
  - https://reviews.llvm.org/D127104 - MachineVerifier: Add test which the verifier incorrectly accepted before
  - Different failures with and without LiveIntervals
- Ambiguously valid MIR - cases which only trigger a verifier error if subregister liveness is enabled
- Chicken and egg failures with LiveIntervals
  - LiveIntervals construction not expecting to handle invalid IR
  - Doesn’t handle unreachable blocks

Remaining Infrastructure Issues

- Targets not fully serializing MachineFunctionInfo
- Some generic fields still not serialized
- LiveIntervals calculation modifies the MIR
  - Sometimes splits new virtual registers
  - Introduces new dead flags
- MachineModuleInfo still has some stateful clutter, but mostly harmless
- -start-before/-stop-after work poorly with failures involving multiple functions

Experiences

- Block reduction pass not yet upstream
  - Highest value reduction
- Managed to make forward progress with difficulty
  - Still much better than manual reduction
  - Manual adjustments to blocks
- Had to run individual reductions to avoid some of the failures
  - Deleting def reduction too slow without liveness consideration
- Behaves like a MIR fuzzer
  - Unsurprisingly, many bugs with undef register handling

Future Needs

- Get work item verification to use LiveIntervals
- Rewrite block reduction to use LiveIntervals to find live registers and get upstream
- CFG simplification pass - not the same as block reduction
    - Fallthrough blocks and trivial successors may matter
- No attempt to handle reducing physical register uses/defs

Outstanding Reviews

- https://reviews.llvm.org/D127107 llvm-reduce: Add reduction pass for MachineBasicBlocks
- https://reviews.llvm.org/D127108 llvm-reduce: Handle reducing blocks using G_BR/G_BRCOND
- https://reviews.llvm.org/D127103 CodeGen: Split out MachineVerifier's liveness tracking

Acknowledgements

- Thanks to Markus Lavin for adding the initial MIR support

Copyright and disclaimer

- The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

- THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION


AMD