Ivan Baev

Controlling Virtual Register Pressure in LLVM Middle-End

QUALCOMM®

Outline

- Motivation
- Related work
- Register pressure background
- LLVM LICM
- LLVM GVN
- Performance results
- Future work: LLVM Inliner
- Summary

Motivation

- When we compared LTO -Ofast vs -Ofast performance we saw 10% degradation in spec2000/crafty benchmark
- Analysis revealed the impact of the following LLVM passes
  - Inliner, LICM, and GVN
- Extra spill code and additional execution time in Evaluate() and Swap()
- Increased register pressure

- This scenario is typical for other compilers too
  - When enabling a new optimization, or increasing the optimization level

Related work

- Register pressure has been a known problem for compiler/performance engineers
  - Mismatch between the infinite number of virtual registers and fixed number of physical registers

- A lot of work to handle register pressure (RP) at machine-level IR: register allocator, scheduler, and related passes
  - Rematerialization (Briggs, 1992)
  - Fighting register pressure in GCC (Makarov, 2004)
  - Prematerialization (Baev, Hank, Gross, 2006)
  - Region-based register allocation (Baev, 2009)
  - Register pressure-aware scheduling (Touati, 2001; Govindarajan, 2003)
  - LLVM register pressure tracking and RP-aware scheduling (Trick, 2012)

Related work (cont.)

- Some work at higher-lever IR: LICM, PRE, and loop transformations
  - Register pressure sensitive redundancy elimination (Gupta, Bodik, 1999)
  - Register pressure guided unroll-and-jam (Ma, Carr, 2008)
  - Model-based framework: an approach for profit-driven optimization (Zhao, Childers, Soffa, 2005)
  - Inlining – most papers acknowledge the problem of register pressure but do not address it directly (Zhao, Amaral, 2003; Chakrabarty, Liu, 2006)
- Handling register pressure at a single place in the compiler – e.g. in register allocator or scheduler – is usually not enough
- This talk will focus on middle-end, target-independent optimizations

Virtual register pressure

- At a given program point
  - the number of overlapping live ranges at that point
- For a basic block(BB)/loop/function
  - the highest register pressure over all program points in the BB/loop/function
- For a BB/loop (in this work)
  - the number of liveins for the BB/loop

- Integer RP, floating-point RP, predicate RP, etc.

- It is an approximation
  - Sources of approximation: register promotion of memory, calls, register pairs, etc.
  - Tradeoff between precision and compile-time, e.g. better precision requires data-flow analysis

Our approach

- Analyze a pass and its components w.r.t. register pressure
  - Study the code, collect statistics
- Add a measure of register pressure to control the component(s) with a high impact

- Allow a component/pass to be invoked multiple times
- Include a comparison with the number of hardware registers of the corresponding type for the target processor

Register pressure analysis of LLVM LICM pass

- Loop-level pass with three components
  - Sinking code
  - Hoisting code
  - Promotion of memory locations

Register pressure analysis of LLVM LICM pass

- Loop-level pass with three components
  - Sinking code – not likely to impact RP (# liveins for the loop)

Register pressure analysis of LLVM LICM pass

- Loop-level pass with three components
  - Sinking code – not likely to substantially impact RP
  - Hoisting code - may impact RP
- Instructions to be hoisted and the impact on RP for the loop

```
%528 = load i64* @rank_mask.1
%527 = load i64* getelementptr inbounds (%struct.CHESS_POSITION.86* @search, i32 0, i32 7)
```

Both instructions increase RP (# liveins) by 1

```
%tobool1418 = icmp ne i64 %and1417, 0   // assume %and1417 is only used in this instruction
```

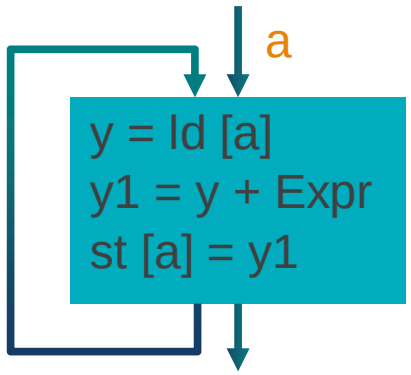
No change in RP

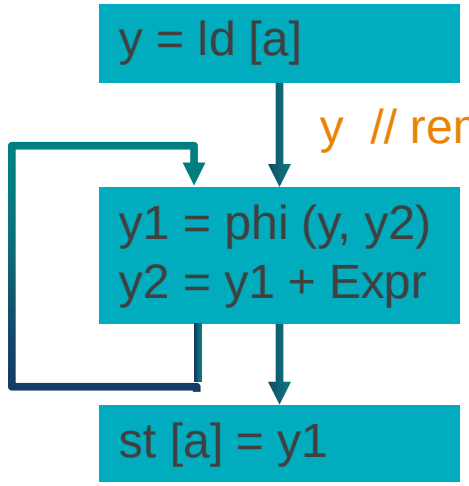
```
%and1417 = and i64 %518, %517   // assume %518 and %517 are only used in this instruction
```

Decrease RP by 1

Register pressure analysis of LLVM LICM pass

- Loop-level pass with three components
  - Sinking code – not likely to impact RP (# liveins for the loop)
  - Hoisting code - may impact RP
  - Promotion of memory locations – not likely to substantially increase RP





Implementation of LICM RP heuristic

```
int MaxLIs // Max number of new liveins allowed for hoisting for the loop
int NumLIs // Current number of new liveins for the loop

estimateRegisterPressure(Loop *L) {
  unsigned MaxLiveIns = TTI->getNumberOfRegisters(false)
  Set LiveIns

  Iterate over all BBs in L
    Iterate over all instructions in BB
      Iterate over source operands in Instruction
        if (Operand is of integer or pointer type)
          if (OperandValue is defined outside L) || (OperandValue is argument or global variable))
            LiveIns.insert(OperandValue)

  NumLIs = 0
  if (LiveIns.cardinality >= MaxLiveIns)
    MaxLIs = 0
  else
    MaxLIs = MaxLiveIns - LiveIns.cardinality
}
// also, provision for user-defined MaxLiveIns (not shown)
```

Implementation of LICM RP heuristic (cont.)

```
bool doesReducePressure(Instruction &I, Loop *L, int &NumLiveInReduce) {
  NumLiveInReduce = -1  // start with -1 due to hoisting I's destination operand

  Iterate over all source operands of I
    if (Operand is of integer or pointer type)
      if ((OperandValue is defined outside L) || (OperandValue is argument or global variable))
        if (OperandValue has one use) NumLiveInReduce++

  if (NumLiveInReduce >= 1) return true
  else return false
}

hoist(Instruction &I) {
  bool ReducePressure = doesReducePressure(I, L, NumLiveInReduced)
  if (NumLIs >= MaxLIs) && !ReducePressure) return  // skip hoisting
  . . .
  hoist I
  NumLIs -= NumLiveInReduced  // keep track of loop's liveins
}
```

Register pressure analysis of LLVM GVN pass

- Function-level pass with two major components
- GVN part
  - processBlock() -> processInstruction()
    - SimplifyInstruction
    - processLoad() -> processNonLocalLoad
    - processBranch
    - processSwitch
    - ProcessOtherInstruction
- PRE part
  - Simple local PRE on diamond control-flow patterns

Implementation GVN RP heuristic

- GVN mostly operates on BB basis

- Estimate RP for the basic block enclosing the load
  - estimateRegisterPressure(BB)
- Estimate RP for the loop enclosing the load
  - estimateRegisterPressure(Loop) // similar to the version in LICM
- Using loop-based RP performs better

- if (estimateRegisterPressure(Loop) >= TTI->getNumberOfRegisters(false))
  skip processing/promoting this load

Performance evaluation of RP heuristics

Benchmark	Speedup with LICM-RP over LTO (%)	Speedup with GVN-RP over LTO (%)	Speedup with both over LTO (%)
ammp	1.7	0.5	1.6
bzip2	-0.8	-2.1	-1.5
crafty	2.5	1.4	4.3
equake	0.4	1.6	1.9
mesa	9.1	3.8	5.7
twolf	3.1	3.1	3.0
vpr	2.2	1.6	2.4

- With QC LLVM compiler - on Nexus4 Android devices, ARMv7, thumb mode
- Good improvements also in ARM mode and for Hexagon processors, for both -Ofast and LTO optimization levels

Controlling RP in Inliner (future work)

- Calculate maxRP for each function traversing the call graph bottom-up
  - At each call site add the callee's RP to the current RP of the caller
- Add RP at call site to the goodness factor (ranking) of the call site
  - In the denominator - as a cost
  - Add extra cost if RP exceeds the number of hardware registers for the target
- Likely no need to update maxRP for a function after inlining any of call sites within the function

Summary

- The register pressure problem will likely to stay
  - Newer generation processors feature more registers, however compiler engineers quickly make the extra registers insufficient
- We presented a general approach and specific heuristics for controlling register pressure in LLVM LICM, GVN, and Inliner passes
  - Will upstream RP patches if there is interest
  - Unroller is another candidate for RP tuning in the middle-end
- Compiler optimizations should be designed to take into account register pressure
  - Simple heuristics are good in many cases
- As a community, continue improving RP in machine-level passes
  - Compute and report the sum of weighted spills when profile information is available

Acknowledgements

Zhaoshi Zheng, Balaram Makam, Yin Ma, Taylor Simpson, QuIC

Any questions?