From Fallbacks to Performance: Towards Better GlobalISel Performance on AArch64 NEON Platforms

Speaker: Madhur Amilkanthwar

In collaboration with:

Dhruv Chawla, Tianyi Guan

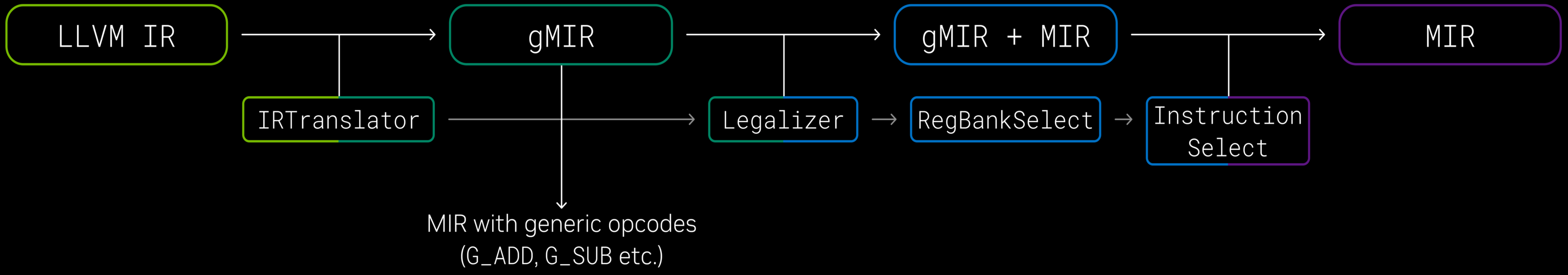
Agenda

- GlobalISel – Quick Introduction and Status
- Our Contributions
- Results
- Next Steps

GlobalISel

- An alternate Instruction Selector that works on linear IR (GMIR, MIR) unlike DAG.
- Translates LLVM IR instructions to MIR in a phased manner.

- Promises compile-time improvements and offers platform to do function-wide optimizations.
- Default instruction selector at -O0 for AArch64 – fallbacks to SDAG for other optimization levels.
- Our experience:
  - TSVC, RajaPerf, SPEC 2017, LLVM Test-suite falls back to SDAG.



Goals

- Our long-term goal is to make GlobalISel the default instruction selector.
- We are taking an empirical approach and trying various known benchmarks with GlobalISel.
- A lot of work to do in supporting basic instructions and patterns.
- But… where do we start?
    - We start with something small and simple which allows us to get the ball rolling.
    - A small benchmark suite which can uncover issues in the framework.
    - Is there one out there which is widely known and small enough? Yes, TSVC!

TSVC and Scope

- TSVC – Test-Suite for Vectorizing Compilers
- Big file containing 152 small computation kernels focusing on loop optimizations.
- Kernel naming convention - `s<number>`

Our Scope:

- We focused on AArch64 Advanced SIMD a.k.a. NEON platform. Let's deal with SVE a bit later!
- Specifically targeting for `–mcpu=grace.`
- Compile with `–O3 -fglobal-isel`

Experimentation Setup:

- Isolated kernels to avoid i-cache and d-cache misses.
- Executed each kernel 5 times and taking minimum.

Status back in Feb 24:
One kernel falls back to SDAG – s317

TSVC s317 Kernel

```
real_t s317(struct args_t * func_args)
{
  // ...
  real_t q;
  for (int nl = 0; nl < 5*iterations; nl++) {
    q = (real_t)1.;
    for (int i = 0; i < LEN_1D/2; i++)
      q *= (real_t).99;
  }
  // ...
  return q;
}
```

- The loop was being vectorized with VF = 4, Unroll Factor = 2.

1

```
%vec.phi = phi <4 x float> [ <1, ...>, %ph ], [ %0, %vec.body ]
%vec.phi20 = phi <4 x float> [ <1, ...>, %ph ], [ %1, %vec.body ]
%0 = fmul fast <4 x float> %vec.phi, <0.99, ...>
%1 = fmul fast <4 x float> %vec.phi20, <0.99, ...>
```

2

```
%1 = phi <8 x float> [ %0, %vector.ph ], [ %3, %vector.body ]
%2 = call <8 x float> @llvm.vector.interleave2.v8f32(<4 x float> <0.99, ...>,
                                                     <4 x float> <0.99, ...>)
%3 = fmul fast <8 x float> %1, %2
```

TSVC s317 Kernel

Fixing the fallback

```
define <8 x i32> @test(<4 x i32> %a, <4 x i32> %b) {
 %c = call <8 x i32> @llvm.vector.interleave2(<4 x i32> %a, <4 x i32> %b)
 ret <8 x i32> %c
}
```

SDAG

GISel

```
t2: v4i32,ch = CopyFromReg t0, Register:v4i32 %0
t4: v4i32,ch = CopyFromReg t0, Register:v4i32 %1
t5: v8i32 = concat_vectors t2, t4
t7: v8i32 = vector_shuffle<0,4,1,5,2,6,3,7> t5,
                              undef:v8i32
```

```
bb.1 (%ir-block.0):
  liveins: $q0, $q1
  %0:_(<4 x s32>) = COPY $q0
  %1:_(<4 x s32>) = COPY $q1
  %2:_(<8 x s32>) = G_SHUFFLE_VECTOR %0:_(<4 x s32>),
    %1:_, shufflemask(0, 4, 1, 5, 2, 6, 3, 7)
```

We added support for lowering the intrinsics to IRTranslator

#85199 [GlobalISel] Add support for interleave and deinterleave intrinsics to IRTranslator

Performance Comparison

No fallback. Everything goes through Global ISel... but what about the performance?

% GISel-SDAG Comparison - Feb'24
25.00%
0.00%
-25.00%
-50.00%
-75.00%
Percentage
GISel Faster
GISel Slower
1
2
s000
s1111
s1112
s1113
s115
s116
s119
s121
s123
s125
s127
s131
s141
s152
s1161
s171
s173
s175
s211
s1213
s1221
s231
s1232
s2233
s241
s243
s1244
s251
s2251
s252
s254
s256
s258
s271
s273
s275
s276
s278
s1279
s2711
s281
s291
s293
s2102
s311
s312
s314
s316
s318
s3110
s3111
s3113
s322
s331
s341
s343
s1351
s353
s1421
s423
s431
s442
s451
s453
s481
s491
s4113
s4115
s4117
va
vas
vpv
vpvtv
vpvpv
vsumr
vbor
Loop

TSVC s1279 Kernel

~65% slower on GlobalISel than SelectionDAG

```
real_t s1279(struct args_t * func_args)
{
  // ...
  for (int nl = 0; nl < iterations; nl++)
    for (int i = 0; i < LEN_1D; i++)
      if (a[i] < (real_t)0.)
        if (b[i] > a[i])
          c[i] += d[i] * e[i];
  // ...
  return calc_checksum(__func__);
}
```

1

The two if-conditions get merged into a single condition, with a select:

```
%0 = fcmp olt <4 x float> %wide.load, zeroinitializer
%1 = fcmp ogt <4 x float> %wide.load39, %wide.load
%2 = select <4 x i1> %0, <4 x i1> %1, <4 x i1> zeroinitializer
```

2

This select gets optimized into an AND after PreLegalizerCombiner, i.e.

```
%15:_(<4 x s1>) = G_FCMP floatpred(olt), %12:_(<4 x s32>), %13:_
%17:_(<4 x s1>) = G_FCMP floatpred(ogt), %16:_(<4 x s32>), %12:_
%72:_(<4 x s1>) = G_FREEZE %17:_
%18:_(<4 x s1>) = G_AND %15:_, %72:_
```

3

```
%164:_(s8) = G_TRUNC %180:_(s32)
%162:_(s8) = G_TRUNC %181:_(s32)
%169:_(s8) = G_FREEZE %168:_
%167:_(s8) = G_FREEZE %166:_
%165:_(s8) = G_FREEZE %164:_
%163:_(s8) = G_FREEZE %162:_
%187:_(s16) = G_ANYEXT %169:_(s8)
%188:_(s16) = G_ANYEXT %167:_(s8)
```

4

The Fixes

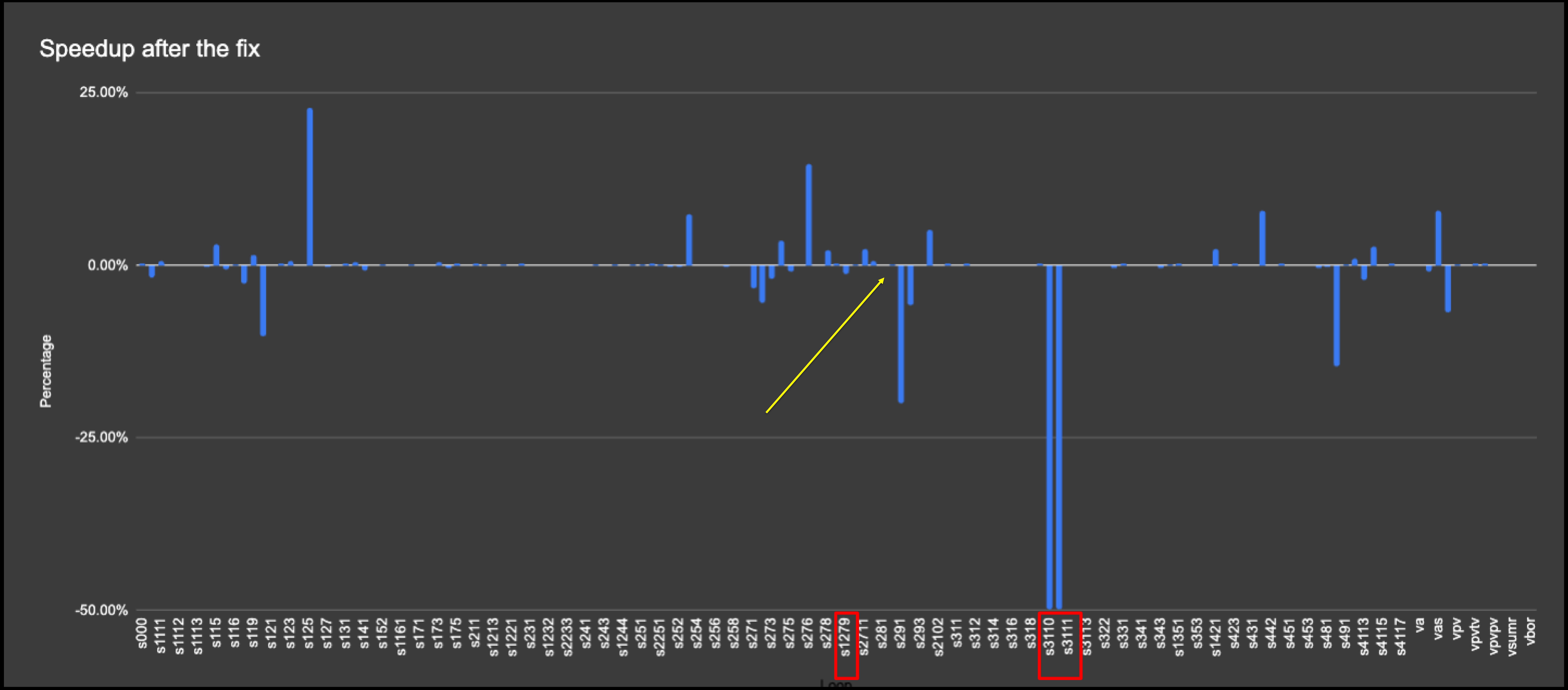
- **The fix adds rules to `AArch64LegalizerInfo.cpp` to avoid expansion.**
  - Issue: Legalizer was scalarizing any `G_FREEZE` not of the type <2 x s64>
  - Fix: legalize `G_FREEZE` like other vector operations

- **Solution:** #88469 `Avoid scalarizing G_FREEZE and G_IMPLICIT_DEF`

- Other fixes:
  - #89017 `[AArch64] Fold COPY(y:gpr, DUP(x:fpr, i)) -> UMOV(y:gpr, x:fpr, i)`
  - #89023 `[GISel][CombinerHelper] Combine op(trunc(x), trunc(y)) -> trunc(op(x, y))`

What do we gain?

- **Performance uplift:** Performance recovered, minor slowdown present.
- Collected at LLVM Commit: `dbc1` (03 June 2024 - After our fixes)



TSVC s3110 Kernel

~50% slower on GlobalISel than SelectionDAG

```c
real_t s3110(struct args_t * func_args)
{
  // ...
  real_t max;
  for (int nl = 0; nl < (...); nl++) {
    max = aa[0][0];
    for (int i = 0; i < LEN_2D; i++)
      for (int j = 0; j < LEN_2D; j++)
        if (aa[i][j] > max)
          max = aa[i][j];
  }
  // ...
}
```

Source Code

1

2

LLVM IR

```llvm
for.body:
  %0 = load float, ptr @aa, align 64
  br (...)

for.body.inner:
  %3     = load float, ptr %<...>, align 4
  %cmp   = fcmp ogt float %3, %max
  %max.2 = select i1 %cmp, float %3, float %max
  br i1 (...)
}
```

TSVC s3110 Kernel

~50% slower on GlobalISel than SelectionDAG

Load of `float` stored into GPR!

Unnecessary copies from GPR inserted

```
# Machine code for function s3110
# ...
bb.3.for.body:
  %0:gpr(s32) = G_LOAD %1:gpr(p0) ::
    (dereferenceable load (s32) from @aa, align 64)

bb.4.for.body.inner.preheader:
  %2:gpr(s32) = G_PHI %0:gpr(s32), %bb.3,
                      %8:fpr(s32), %bb.<...>
  G_BR (...)

bb.7.for.body.inner:
  %3:gpr(s32)  = G_PHI %2:gpr(s32), %bb.4,
                       %8:fpr(s32), %bb.7
  %6:fpr(s32)  = G_LOAD %20:gpr(p0) ::
                     (load (s32) from %ir.<...>)
  %4:fpr(s32) = COPY %3:gpr(s32)
  %7:gpr(s32) = G_FCMP floatpred(ogt), %6:fpr(s32), %4:fpr
  %5:fpr(s32) = COPY %3:gpr(s32)
  %8:fpr(s32)  = G_SELECT %7:gpr(s32), %6:fpr, %5:fpr
  G_BR (...)
```

Issue:

`RegBankSelect` phase failed to assign optimal register banks to operands.

It did not analyze uses of G_LOAD beyond PHI nodes and assigned GPR bank which resulted in unnecessary copies.

The fix:

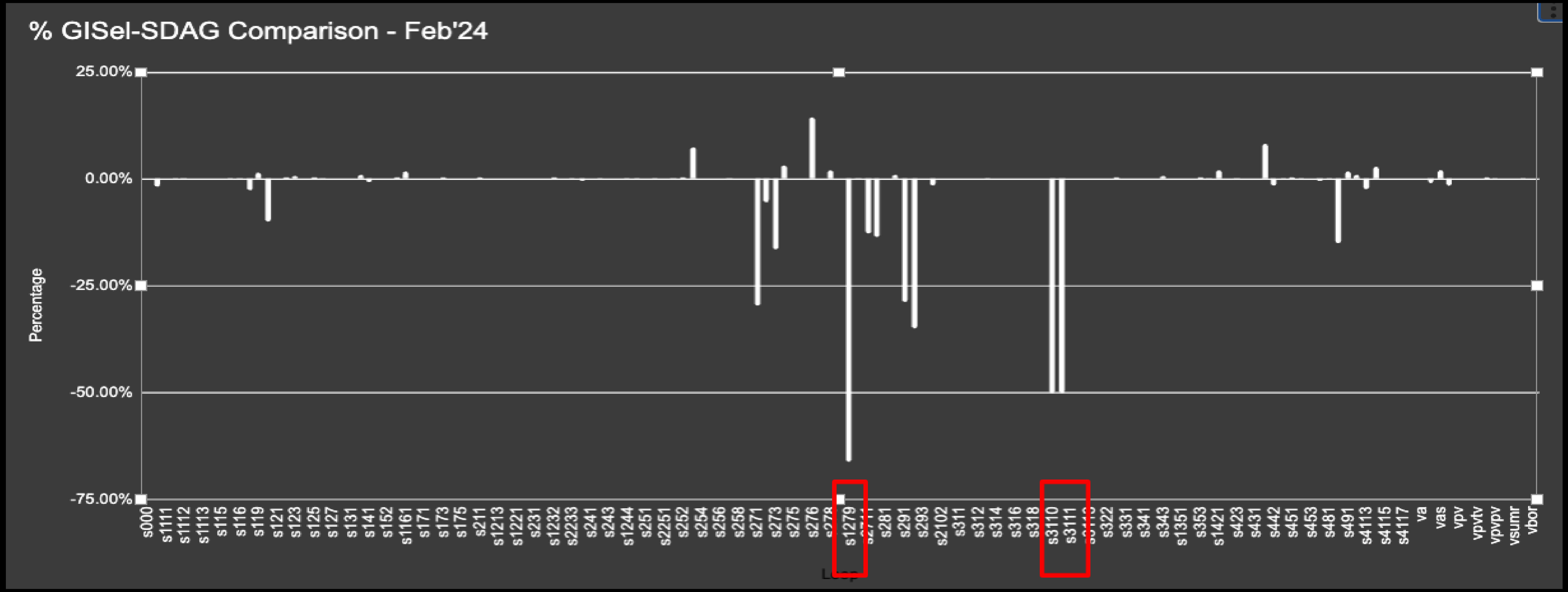
Analyze uses of G_LOAD in PHI node recursively.

If the def of PHI is used by instructions that only uses FP operands, then assign FPR to the def of G_LOAD.

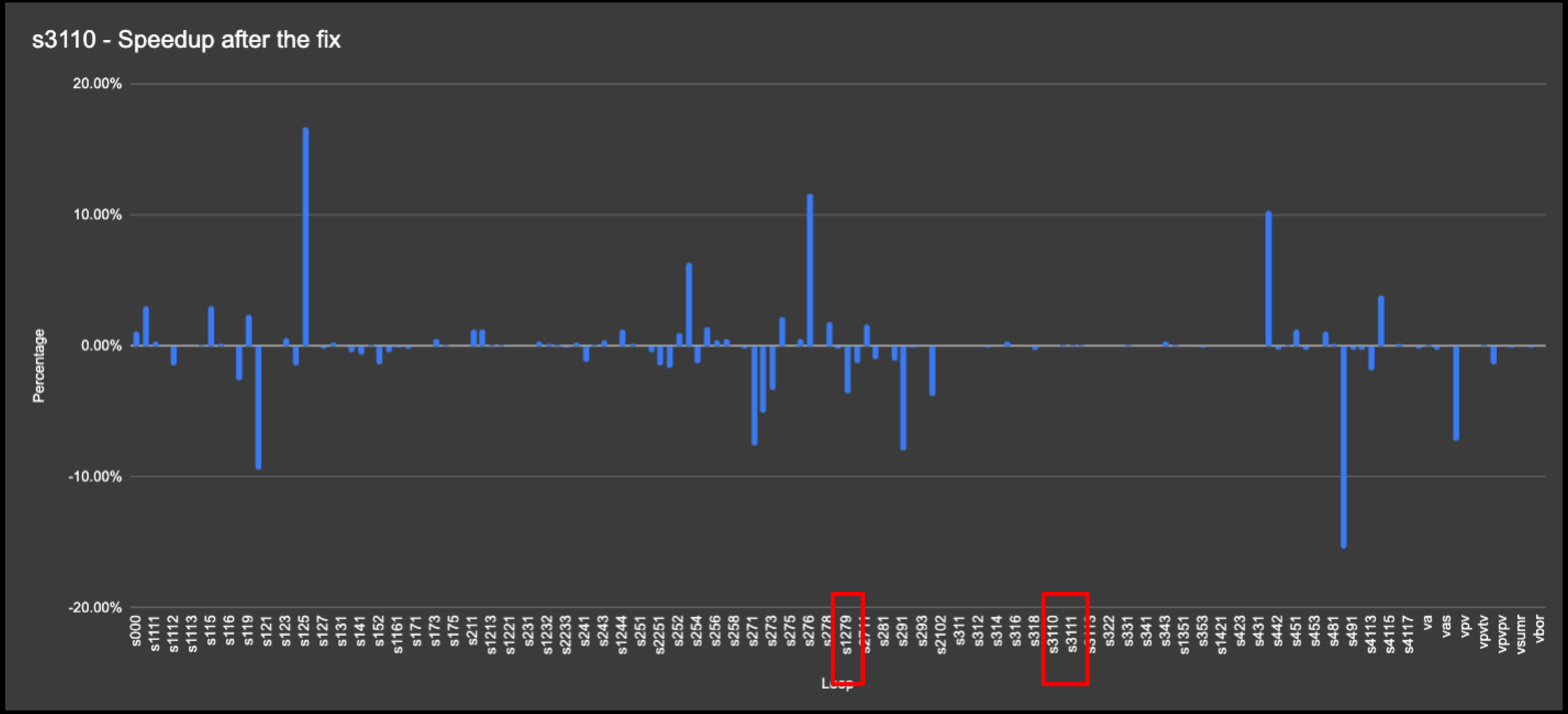
The patch: #94618

Finally... where do we stand?

Before



After



Conclusions

Achievements:

- Fallback free TSVC at -O3 level for Advanced SIMD (Neon)
- Compile-time neutral performance parity of GlobalIsel with Selection DAG for TSVC.

Future Work:

- Ongoing SVE support.
- Benchmark and workload analysis
  - SPEC 2017, RAJAPerf, LLVM Test-suite and further.
- GlobalISel by default is a long road
  - Patches welcome, come join us!

Current Status

Benchmark	# fallbacks	Notes
SPEC 2017, intrate	52	4 out of 9 benchmarks compile without fallbacks to SDAG.
RAJAPerf	6	G_EXT, G_AND and G_PTRTOINT
LLVM Test-suite	1029	Inline asm, G_EXT, G_FREEZE and many more are missing support.

our patch to support varargs in InstructionSelect phase reduced many fallbacks.

Thank you for attending!