Extending LLVM optimization repertoire to build a highly optimizing compiler

Ehsan Amiri, Bryan Chan
Huawei Technologies Canada
ehsan.amiri@huawei.com, bryan.chan@huawei.com

www.huawei.com

Introduction

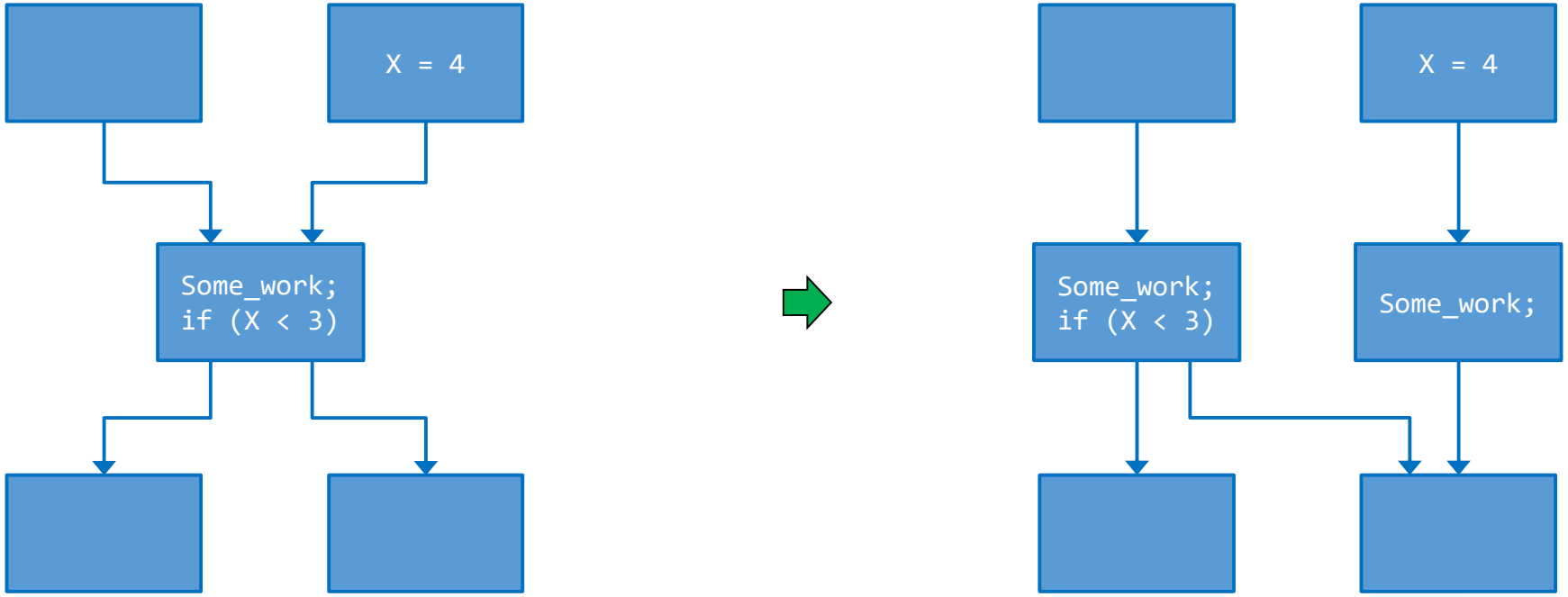
- We report some of the optimizations developed for Bisheng Compiler at Huawei.
- Focus on work that is either contributed to the community or is likely to be contributed.
- Several colleagues at Huawei have contributed to the work reported here:

  - Alexey Zhikhartsev
  - Congzhe Cao
  - Justin Kreiner
  - Sagar Kulkarni
  - Shruthi Ashwathnarayan
  - Yibo (Nigel) Yu
  - Zhongduo (Jimmy) Lin

Agenda

- **DFA Jump Threading**
- Speculative Instruction Combining
- Loop Interchange
- ARMv9-A Scalable Matrix Extension

Jump Threading



Limitations of the existing jump threading

- Iterative approach:
  - Look at a couple of BBs, catch the opportunity, iterate.

```
/// processBlock - If there are any predecessors whose control can be threaded
/// through to a successor, transform them now.
bool JumpThreadingPass::processBlock(BasicBlock *BB) {
```

```
static cl::opt<bool> ThreadAcrossLoopHeaders(
    "jump-threading-across-loop-headers",
    cl::desc("Allow JumpThreading to thread across loop headers, for testing"),
    cl::init(false), cl::Hidden);
```

- Don’t thread across loop headers.

```
// Don't alter Loop headers and latches to ensure another pass can
// detect and transform nested loops later.
!LoopHeaders.count(&BB) && !LoopHeaders.count(Succ) &&
TryToSimplifyUncondBranchFromEmptyBlock(&BB, DTU)) {
```

Limitations of the existing jump threading

- Iterative approach:
  - Look at a couple of BBs, catch the opportunity, iterate.

```
/// processBlock - If there are any predecessors whose control can be threaded
/// through to a successor, transform them now.
bool JumpThreadingPass::processBlock(BasicBlock *BB) {
```

```
static cl::opt<bool> ThreadAcrossLoopHeaders(
    "jump-threading-across-loop-headers",
    cl::desc("Allow JumpThreading to thread across loop headers, for testing"),
    cl::init(false), cl::Hidden);
```

- Don’t thread across loop headers.

```
// Don't alter Loop headers and latches to ensure another pass can
// detect and transform nested loops later.
!LoopHeaders.count(&BB) && !LoopHeaders.count(Succ) &&
TryToSimplifyUncondBranchFromEmptyBlock(&BB, DTU)) {
```

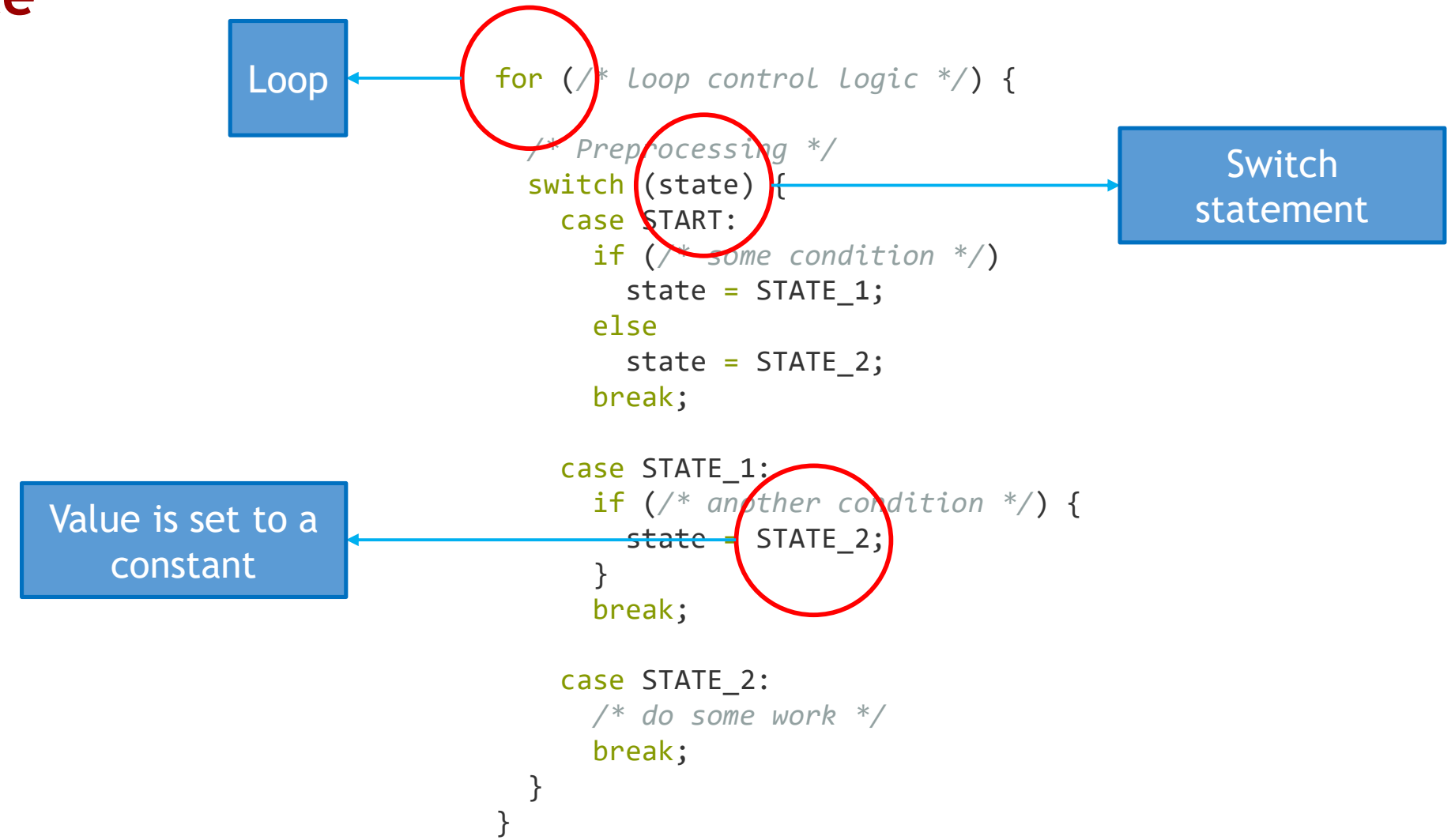
- Impact: Missing opportunities. Notable example is CoreMark.

Not a jump threading for CoreMark

- Let’s review the problem
  - Have a missed performance opportunity.
  - Rewriting jump threading is a much bigger project.
  - Extending current jump threading is not an option.

  - Let’s start from the immediate case that we need to solve.
  - Try to design a general/generalizable solution.

Example



DFA implementation is a typical example, but the algorithm is not limited to DFA implementation.

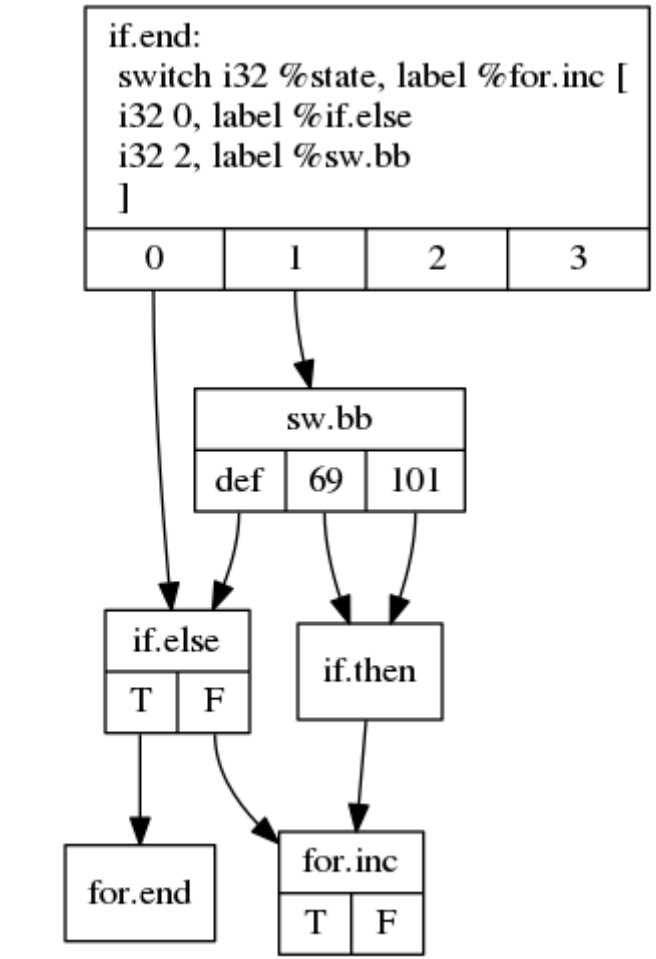
Basic idea

```
/* Preprocessing */
switch (state) {
  case START:
    if (/* some condition */)
      state = STATE_1;
    else
      state = STATE_2;
    break;

  case STATE_1:
    if (/* another condition */) {
      state = STATE_2;
      /* Update IV. Compare bound and branch */
      /* Preprocessing */
      /* Goto 'case STATE_2:' */
    }
    break;

  case STATE_2:
    /* do some work */
    break;
}
```

What is the problem to solve?

- We need to decide:
  - Which basic blocks need to be cloned.
    - And how many clones of each basic block we need.
  - How to connect cloned basic blocks.
  - Whether the transformation is profitable.

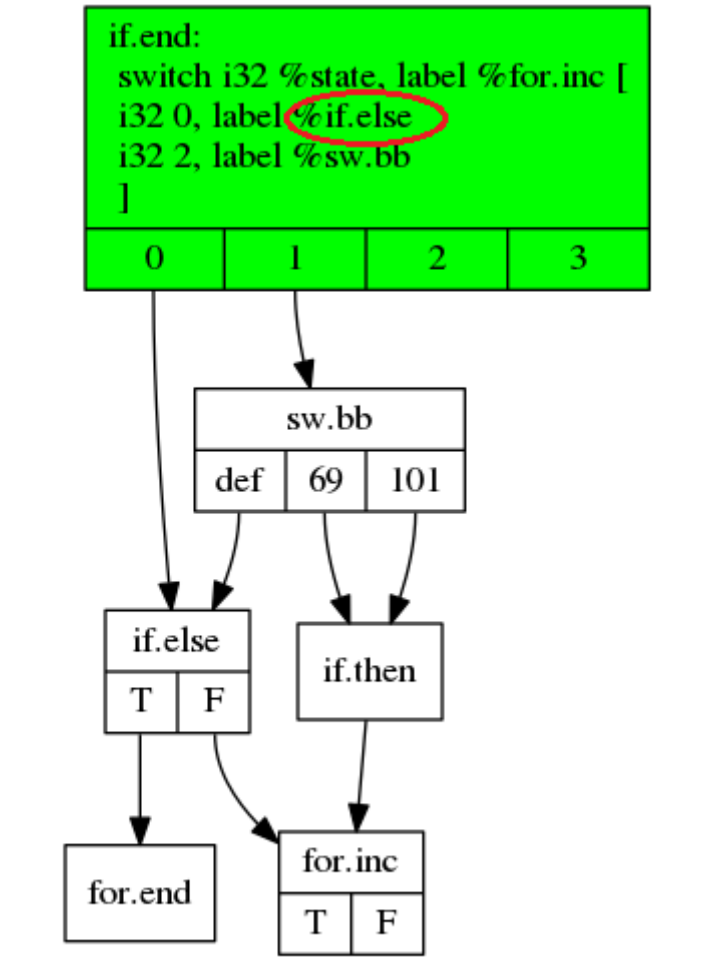
Analysis algorithm – an example



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

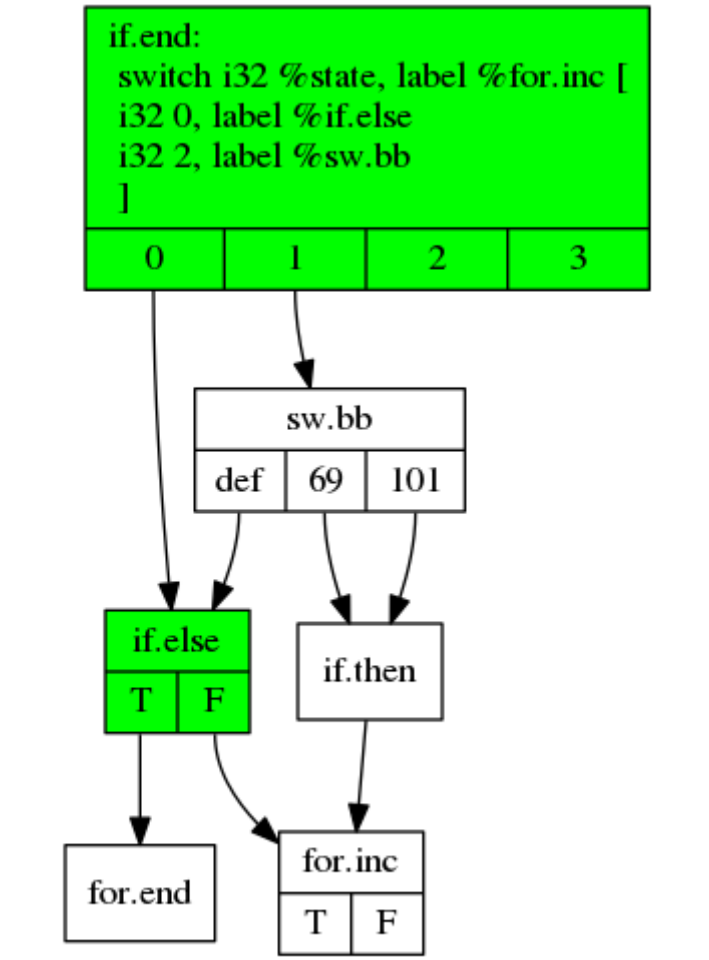
- <if.end>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

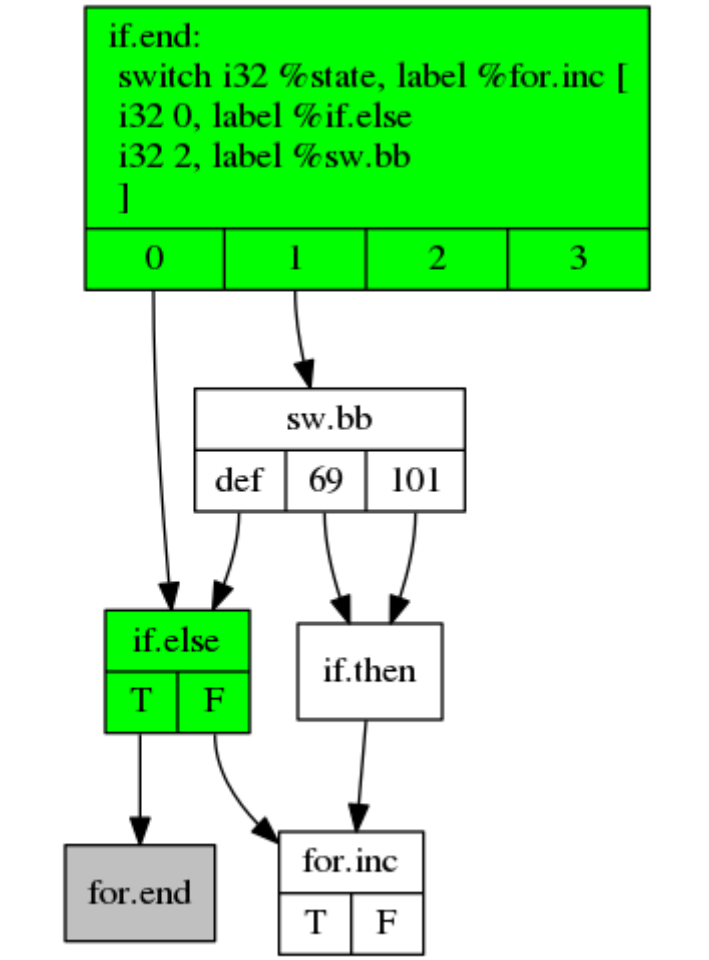
- <if.end, if.else>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

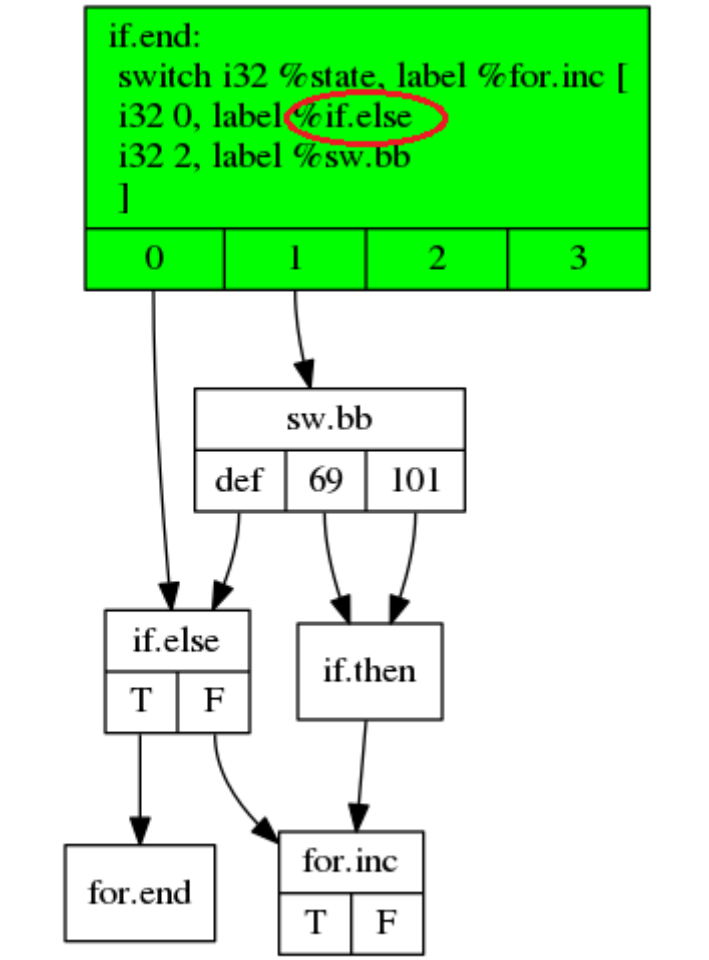
- <if.end, if.else, for.end>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

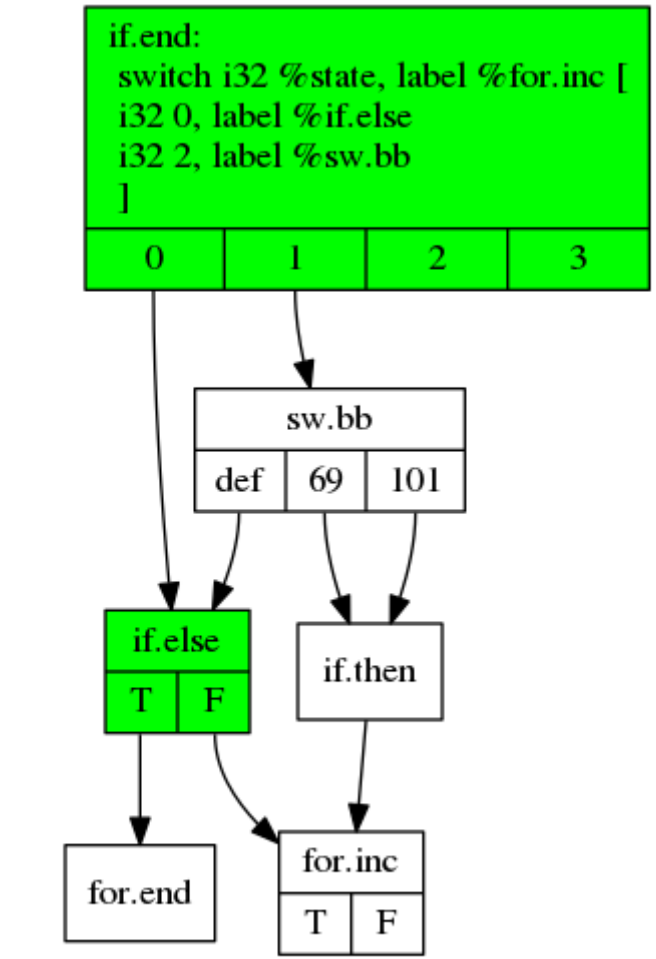
- <if.end, if.else, for.end>
- <if.end>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

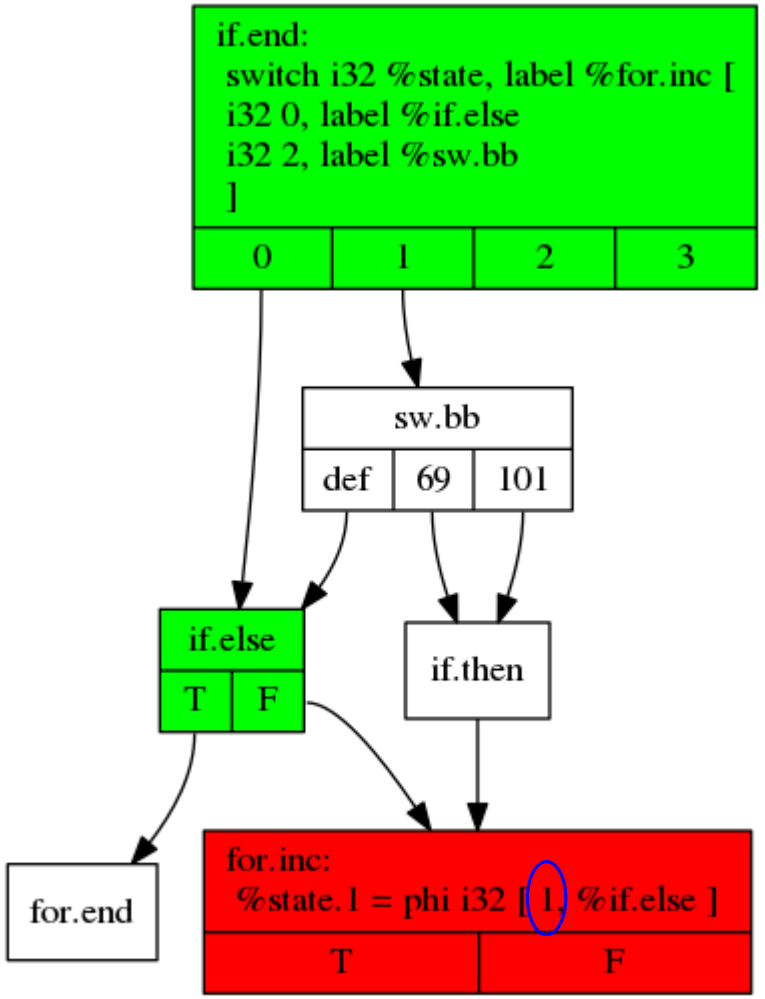
- <if.end, if.else, for.end>
- <if.end, if.else>



Toy example to demonstrate analysis algorithm

Analysis algorithm - an example

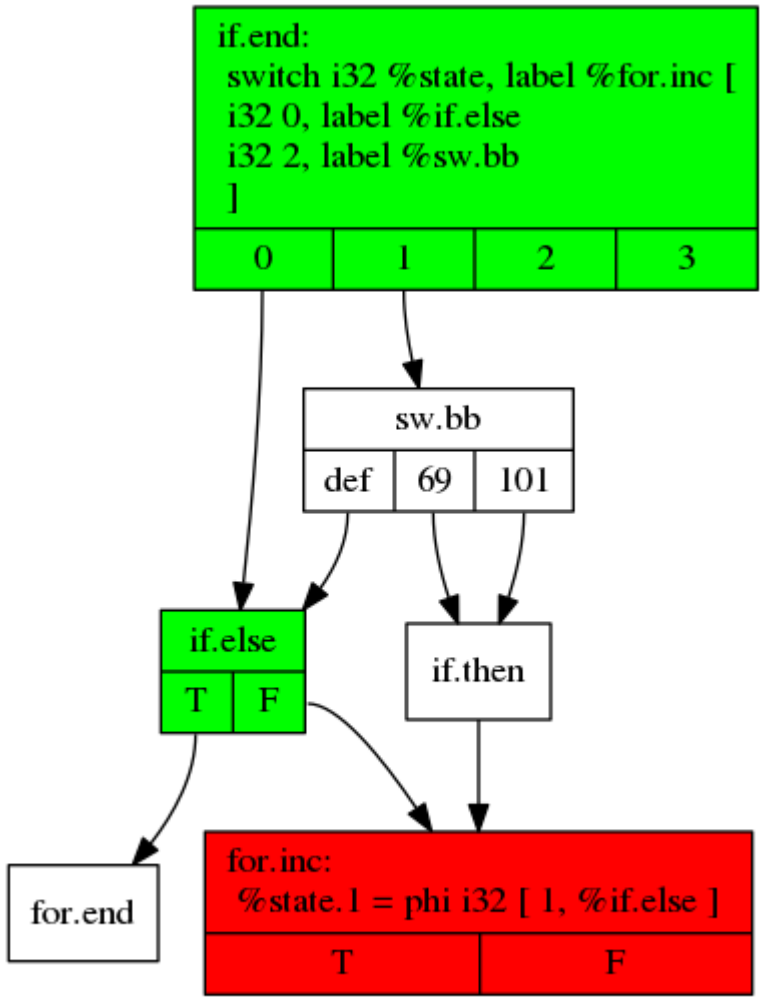
- <if.end, if.else, for.end>
- <if.end, if.else, for.inc>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

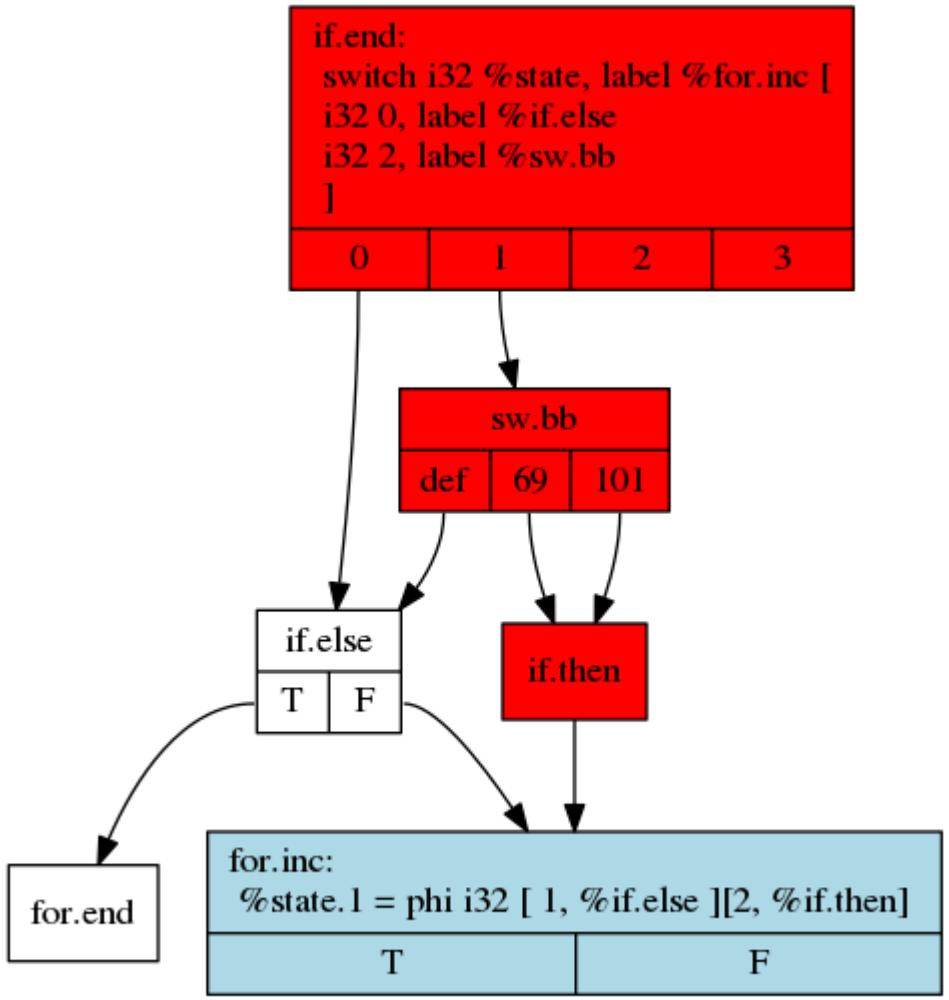
- <if.end, if.else, for.end>
- <if.end, if.else, for.inc, if.end>



Toy example to demonstrate analysis algorithm

Analysis algorithm - an example

- <if.end, if.else, for.end>
- <if.end, if.else, for.inc, if.end>
- <if.end, sw.bb, if.then, for.inc, if.end>



Toy example to demonstrate analysis algorithm

Analysis algorithm – an example

- <if.end, if.else, for.end>
- <if.end, if.else, for.inc, if.end>
- <if.end, sw.bb, if.then, for.inc, if.end>

- One copy of each BB, for each pair of (BB, color).
- For our example: if.end, if.end, if.end means three copies of “if.end”.
- Adjacent nodes in one path should be connected to each other.
- Irreducible control flow can be detected if needed.
- A fairly simple cost model: weighing code size increase vs. #conditional branches eliminated.
- CodeGen relies on this information and heavily utilizes **SSAUpdaterBulk**.
- https://reviews.llvm.org/D99205

Statistics and compile time

Number of transformed switch statements:

```
CTMark/kimwitu++/kimwl.stats:    "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/LzmaEnc.stats:       "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/DeflateDecoder.stats:        "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/GzHandler.stats:     "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/CabHandler.stats:    "dfa-jump-threading.NumTransforms": 3
CTMark/7zip/ShrinkDecoder.stats:         "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/Update.stats:        "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/List.stats: "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/Extract.stats:       "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/ZipHandlerOut.stats:         "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/TarHandler.stats:    "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/7zUpdate.stats:      "dfa-jump-threading.NumTransforms": 2
CTMark/7zip/XzDec.stats:         "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/OpenArchive.stats:   "dfa-jump-threading.NumTransforms": 1
CTMark/7zip/BwtSort.stats:       "dfa-jump-threading.NumTransforms": 1
CTMark/ClamAV/libclamav_untar.stats:     "dfa-jump-threading.NumTransforms": 1
CTMark/ClamAV/libclamav_message.stats:   "dfa-jump-threading.NumTransforms": 1
CTMark/sqlite3/sqlite3.stats:    "dfa-jump-threading.NumTransforms": 10
CTMark/SPASS/iascanner.stats:    "dfa-jump-threading.NumTransforms": 1
CTMark/SPASS/dfgscanner.stats:   "dfa-jump-threading.NumTransforms": 1
CTMark/consumer-typeset/z36.stats:       "dfa-jump-threading.NumTransforms": 1
CTMark/consumer-typeset/z49.stats:       "dfa-jump-threading.NumTransforms": 1
CTMark/consumer-typeset/z38.stats:       "dfa-jump-threading.NumTransforms": 1
```

NewPM-O3:

Benchmark	Old	New
kimwitu++	60426M	60478M (+0.09%)
sqlite3	52644M	52802M (+0.30%)
consumer-typeset	47826M	48036M (+0.44%)
Bullet	118033M	117991M (-0.04%)
tramp3d-v4	104978M	105086M (+0.10%)
mafft	45849M	45833M (-0.03%)
ClamAV	71376M	71413M (+0.05%)
lencod	86159M	86165M (+0.01%)
SPASS	57941M	57987M (+0.08%)
7zip	172904M	173201M (+0.17%)
geomean	74581M	74669M (+0.12%)

Agenda

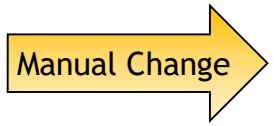
- DFA Jump Threading
- **Speculative Instruction Combining**
- Loop Interchange
- ARMv9-A Scalable Matrix Extension

Speculative instruction combining - motivation

```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
  }
  return temp1 * weight + temp2 * (1 - weight);
}
```

Speculative instruction combining - motivation

```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
  }
  return temp1 * weight + temp2 * (1 - weight);
}
```




```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
    return temp1 * weight + temp2 * (1 - weight);
  }
  return temp1 * weight + temp2 * (1 - weight);
}
```

Speculative instruction combining - motivation

```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
  }
  return temp1 * weight + temp2 * (1 - weight);
}
```

Manual Change

```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
    return temp1 * weight + temp2 * (1 - weight);
  }
  return temp1 * weight + temp2 * (1 - weight);
}
```

Inst Combine

```
float f1(float *vals, int idx, float weight) {

  float temp1 = vals[idx];
  float temp2 = vals[idx];

  if (idx < 10) {
    temp1 += vals[idx+1];
    return temp1 * weight + temp2 * (1 - weight);
  }
  return temp1;
}
```

How to do it automatically? A look at the IR

```
; Function Attrs: norecurse nounwind readonly uwtable willreturn
define dso_local float @f1(float* nocapture readonly %vals, i32 %idx, float %weight) local_unnamed_addr #0 {
entry:
  %idxprom = sext i32 %idx to i64
  %arrayidx = getelementptr inbounds float, float* %vals, i64 %idxprom
  %0 = load float, float* %arrayidx, align 4, !tbaa !6
  %cmp = icmp slt i32 %idx, 10
  br i1 %cmp, label %if.then, label %if.end

if.then:                                          ; preds = %entry
  %add = add nsw i32 %idx, 1
  %idxprom3 = sext i32 %add to i64
  %arrayidx4 = getelementptr inbounds float, float* %vals, i64 %idxprom3
  %1 = load float, float* %arrayidx4, align 4, !tbaa !6
  %add5 = fadd fast float %1, %0
  br label %if.end

if.end:                                           ; preds = %if.then, %entry
  %temp1.0 = phi float [ %add5, %if.then ], [ %0, %entry ]
  %2 = fsub fast float %temp1.0, %0
  %3 = fmul fast float %2, %weight
  %add7 = fadd fast float %3, %0
  ret float %add7
}
```

A look at the IR

- We want to focus on this BB

```
if.end:                                           ; preds = %if.then, %entry
  %temp1.0 = phi float [ %add5, %if.then ], [ %0, %entry ]
  %2 = fsub fast float %temp1.0, %0
  %3 = fmul fast float %2, %weight
  %add7 = fadd fast float %3, %0
  ret float %add7
```

A look at the IR

- Clone it to 2 BBs, one for each incoming value of the phi node.

```
if.end:                                           ; preds = %if.then
  %temp1.0 = phi float [ %add5, %if.then ]
  %2 = fsub fast float %temp1.0, %0
  %3 = fmul fast float %2, %weight
  %add7 = fadd fast float %3, %0
  ret float %add7
```

```
if.end:                                           ; preds = %entry
  %temp1.0 = phi float [ %0, %entry ]
  %2 = fsub fast float %temp1.0, %0
  %3 = fmul fast float %2, %weight
  %add7 = fadd fast float %3, %0
  ret float %add7
```

High level idea

- Clone the IR. Use known value/property of some variable to optimize the code.
  - Call site splitting does this for call instructions.
  - Loop peeling does this for loops.
  - Function specialization performs this for an entire function.

Challenges and the current implementation

- What should be the scope of optimization? For now, we chose one Basic Block.
- InstCombine can follow use-def chains to anywhere in the function. So the Basic Block has to be isolated.
- Currently we isolate the BB into a separate function and run InstCombine on it. We keep a clone if there is a good enough reduction in the number of instructions.

Agenda

- DFA Jump Threading
- Speculative Instruction Combining
- **Loop Interchange**
- ARMv9-A Scalable Matrix Extension

Loop interchange

- Disabled by default in the pipeline.
- The pass is not functionally stable and has fairly significant limitations.
- We have improved this pass to make it functionally stable and more aggressive.
- Today it is turned on for some of our workloads with positive impact on the performance.

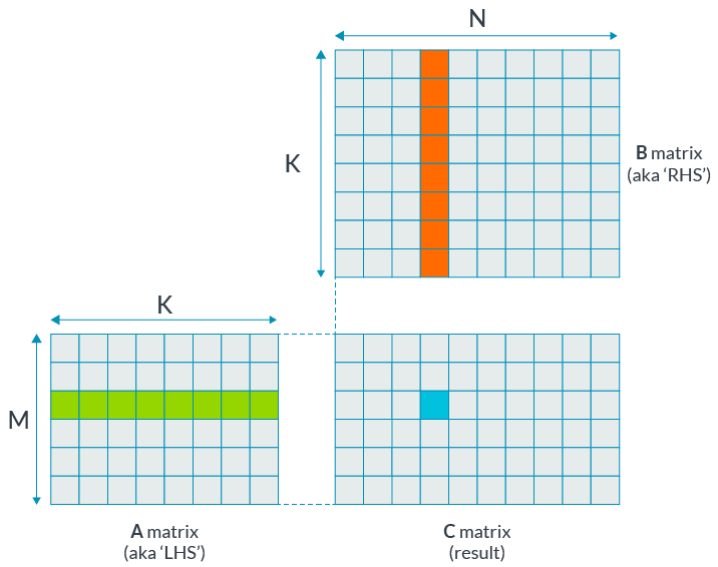
  We expect to be able to turn it on by default soon.
- Part of the work upstreamed. More patches will be posted soon.
- Will provide a more detailed report on this in near future.

Functional Problems	Missed Opportunities	Contributed Patches
Detection of perfect loop nest	Multiple outer/inner induction vars	https://reviews.llvm.org/D98263
Legality check for triangular loops	Better ordering of loops in a loop nest	https://reviews.llvm.org/D101305
Handling LCSSA phis	Better coverage of reduction loops • Under Complex control flow • Floating point • Multi level loop nests.	https://reviews.llvm.org/D102300
Transformation of reduction phis		https://reviews.llvm.org/D98475
Connection of new outer latch to header		https://reviews.llvm.org/D100792
		https://reviews.llvm.org/D102743

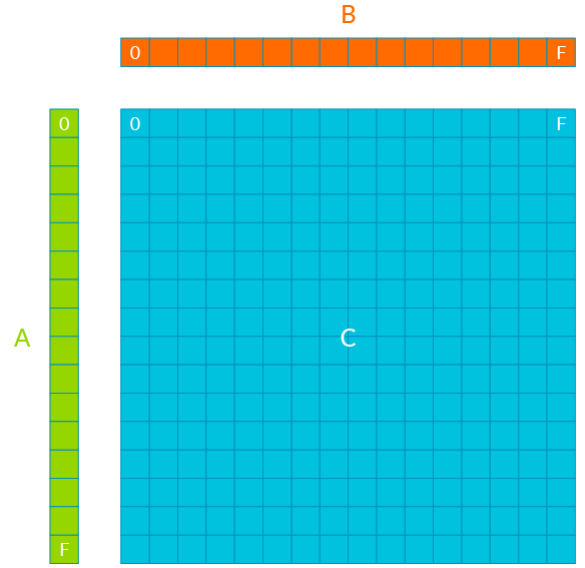
Agenda

- DFA Jump Threading
- Speculative Instruction Combining
- Loop Interchange
- **ARMv9-A Scalable Matrix Extension**

Clang Support for ARMv9-A Scalable Matrix Extension



vs.




```
// matrix of inner products (scalars)
for m in 0..M-1
     for n in 0..N-1
          C[m, n] = 0;
          for k in 0..K-1
               C[m, n] += A[m, k] * B[k, n]
```

```
// sum of matrix outer products
C[0..m, 0..n] = 0;
for k in 0..K-1
     C[0..m, 0..n] += A[0..m, k] ⊗ B[k, 0..n]
```

- Matrix outer product instructions: fewer loops, better memory access patterns
- Project Goal: Clang intrinsics for SME programming in C

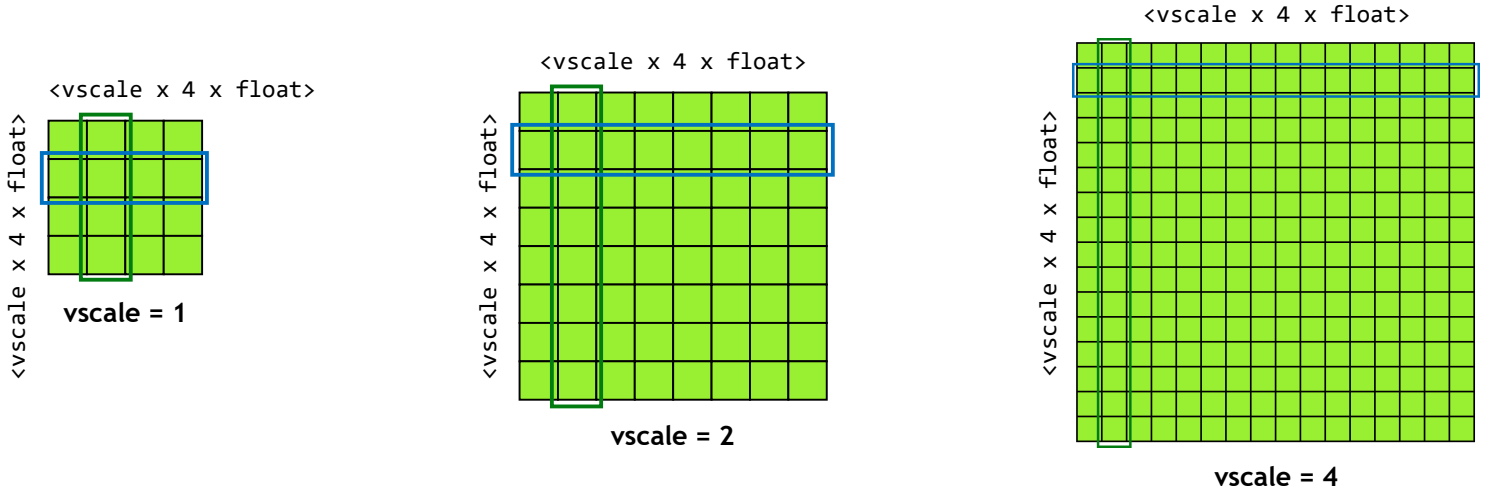
https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/scalable-matrix-extension-armv9-a-architecture

Ingredients

- Clang Intrinsics
- **LLVM IR Extension**
- Instruction Selection and Lowering
- Stack Management
- Register Allocation
- MC Support

LLVM IR Extension

- Every row and column in a scalable matrix register ("tile") is a scalable vector
  - Can be moved to or from a SVE register



- Therefore number of elements in a tile is proportional to (vscale * vscale)
  - Cannot be represented with a `ScalableVectorType`

- Need a new class of IR types

LLVM IR Extension: Design #1

- Proposal: Add a new implementation-defined scaling factor rscale, used together with vscale
- Matrix types would look like:
  - `<rscale x vscale x 16 x i8>`, `<rscale x vscale x 8 x half>`, `<rscale x vscale x 4 x float>`, etc.
- Pros and Cons
  - Rectangular (non-square) scalable matrices without IR-level predication ✓
  - Construction very different from existing scalable vector types ✗
  - Rows and columns are not equally scalable vectors ✗
  - Unnecessary complexity for AArch64 ✗

LLVM IR Extension: Design #2

- Consider a scalable vector type, which multiplies a fixed vector type with a scaling factor
  - *vscale* * `<16 x i8>` = `<vscale x 16 x i8>`
- Think of a scalable matrix type as multiplying a basic matrix type with a scaling factor
  - Example: A basic matrix of bytes consisting of 16 * `<16 x i8>` vectors = `<256 x i8>`
  - *mscale* * `<256 x i8>` = `<mscale x 256 x i8>`

LLVM IR Extension: Design #2

- Consider a scalable vector type, which multiplies a fixed vector type with a scaling factor
  - *vscale* * `<16 x i8>` = `<vscale x 16 x i8>`
- Think of a scalable matrix type as multiplying a basic matrix type with a scaling factor
  - Example: A basic matrix of bytes consisting of 16 * `<16 x i8>` vectors = `<256 x i8>`
  - *mscale* * `<256 x i8>` = `<mscale x 256 x i8>`
- Another example

FixedVectorType	**ScalableVectorType**	**"Basic Matrix"**	**ScalableMatrixType**
`<4 x float>`	`<vscale x 4 x float>`	`<16 x float>`	`<mscale x 16 x float>`

LLVM IR Extension: Design #2

- Think of a scalable matrix type as multiplying a basic matrix size with a scaling factor

ValueType	FixedVectorType	ScalableVectorType	Elements Per Row /Column	ScalableMatrixType
i8	<16 x i8>	<vscale x 16 x i8>	16 * vscale	<mscale x 256 x i8>
i16	<8 x i16>	<vscale x 8 x i16>	8 * vscale	<mscale x 64 x i16>
i32	<4 x i32>	<vscale x 4 x i32>	4 * vscale	<mscale x 16 x i32>
i64	<2 x i64>	<vscale x 2 x i64>	2 * vscale	<mscale x 4 x i64>
i128	<1 x i128>	<vscale x 1 x i128>	1 * vscale	<mscale x 1 x i128>
half	<8 x half>	<vscale x 8 x half>	8 * vscale	<mscale x 64 x half>
float	<4 x float>	<vscale x 4 x float>	4 * vscale	<mscale x 16 x float>
double	<2 x double>	<vscale x 2 x double>	2 * vscale	<mscale x 4 x double>

- Pros and Cons
  - Similarity to existing scalable vector types: less effort ✓
  - Rows and columns are both scalable vectors with the same vscale ✓
  - Requires explicit predication to handle non-square matrices ✗

On-going Work

- Can compile Clang intrinsics into LLVM IR, then into ARMv9-A SME instructions!
- Arm has upstreamed MC support for SME instructions this summer
- Rebase prototype on main branch and refine
- RFC: LLVM IR extension for scalable matrix types and intrinsics
- TLX: New proposal for Tensor LLVM Extensions

```
if.end40:                                         ; preds = %if.end40.lr.ph, %if.end40
  %11 = call <vscale x 2 x i1> @llvm.aarch64.sve.convert.from.svbool.nxv2i1(<vscale x 16 x i1> %2)
  %12 = call <vscale x 2 x i1> @llvm.aarch64.sve.convert.from.svbool.nxv2i1(<vscale x 16 x i1> %4)
  %14 = call <vscale x 2 x double> @llvm.aarch64.sme.mova.vec.col.nxv2f64.mxv4f64(<vscale x 2 x i1> %11, <mscale x 4 x double> %zb.0.lcssa, i32 %conv41, i32 0)
  %mul44 = mul i64 %add35156, %conv11
  %add.ptr46 = getelementptr inbounds double, double* %add.ptr45, i64 %mul44
  %15 = call <vscale x 2 x double> @llvm.aarch64.sve.ld1.nxv2f64(<vscale x 2 x i1> %12, double* %add.ptr46)
  %16 = call <mscale x 4 x double> @llvm.aarch64.sme.fmopa.mxv4f64.nxv2f64(<vscale x 2 x i1> %11, <vscale x 2 x i1> %12, <mscale x 4 x double> %za.1154,
                                                                           <vscale x 2 x double> %14, <vscale x 2 x double> %15)
```

Example of scalable matrix IR

Thank you

www.huawei.com

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.