Three years experience with a tree-like shader IR

Ian Romanick
ian.d.romanick@intel.com
1-February-2014

Agenda

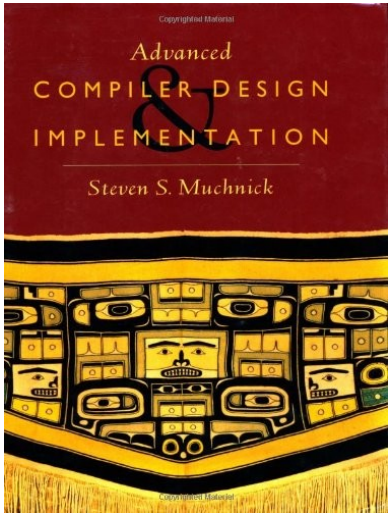
- Background
- Problems with current IR
- Steps towards a better IR

Background

- In 2010, Mesa's GLSL compiler was in dire shape

Background

- Complete compiler re-write
  - “Advanced Compiler Design and Implementation” by Steven Muchnick
  - “A Retargetable C Compiler: Design and Implementation” by David Hanson

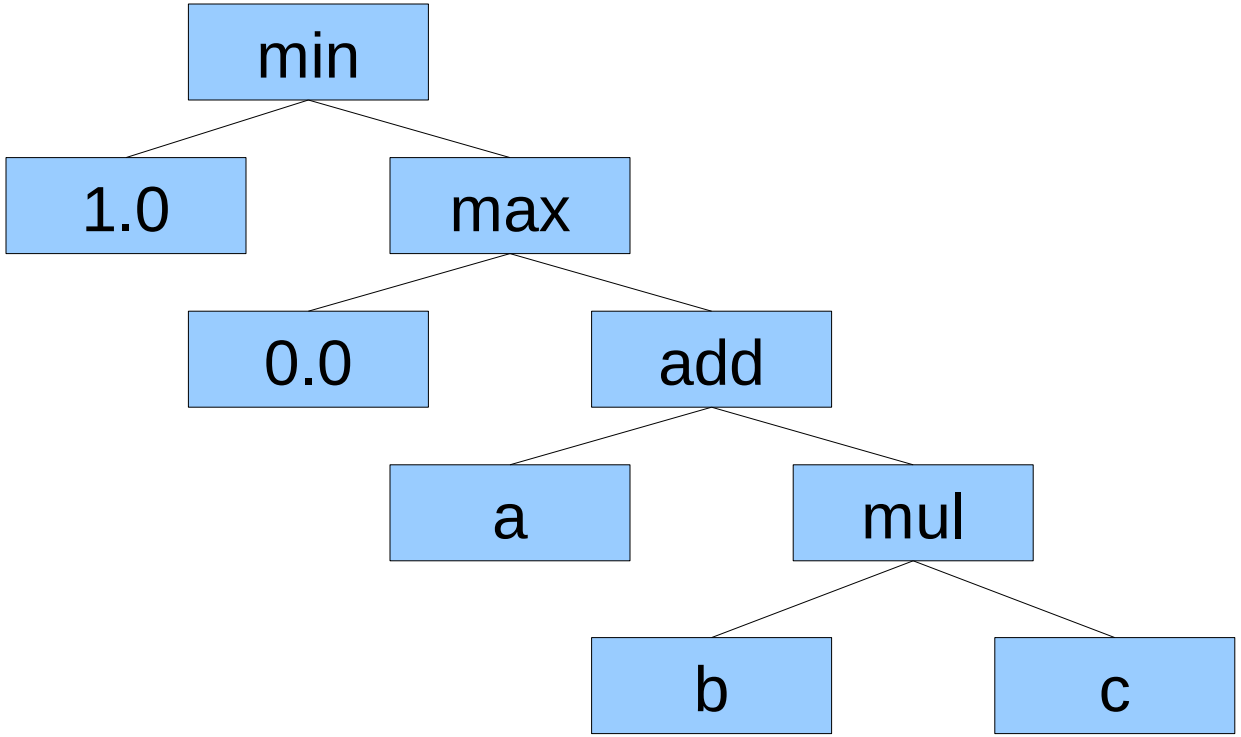


Tree-like IR – Expressions

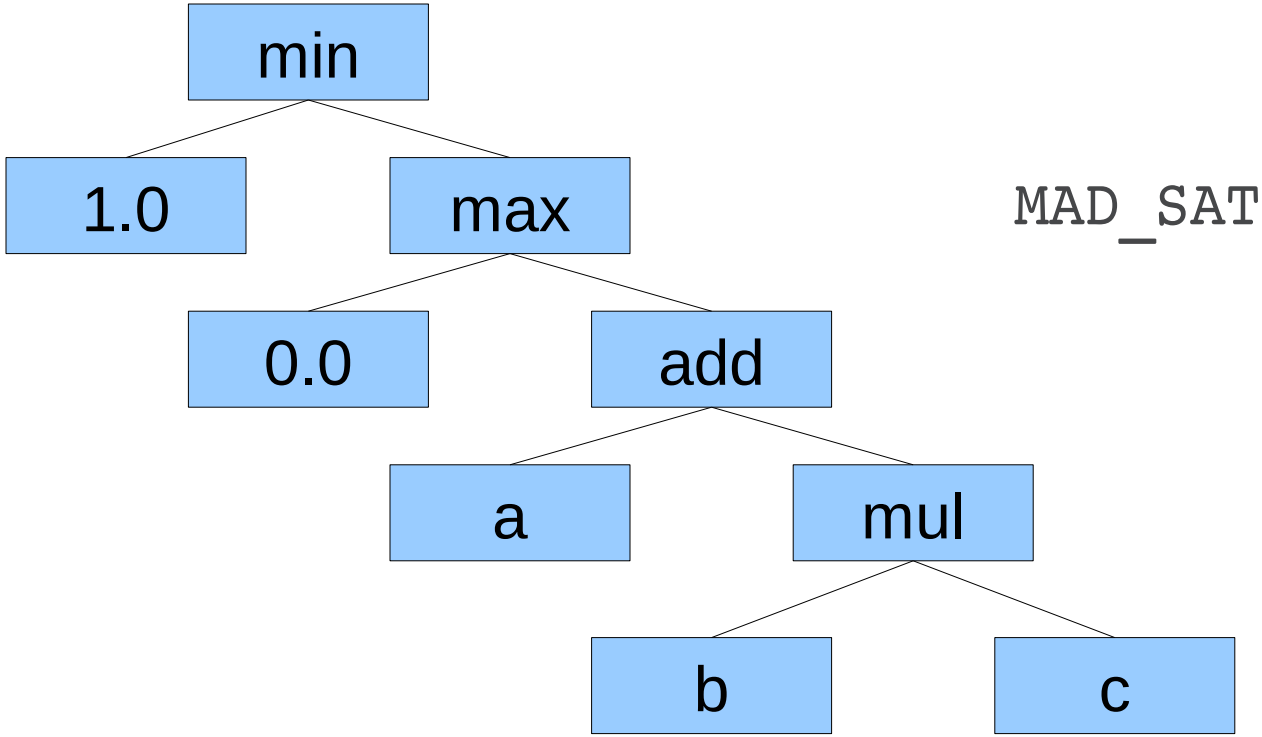
```
class ir_rvalue : public ir_instruction {
   const struct glsl_type *type;
};

class ir_expression : public ir_rvalue {
   ir_expression_operation operation;
   ir_rvalue *operands[4];
};
```


Tree-like IR – Expressions

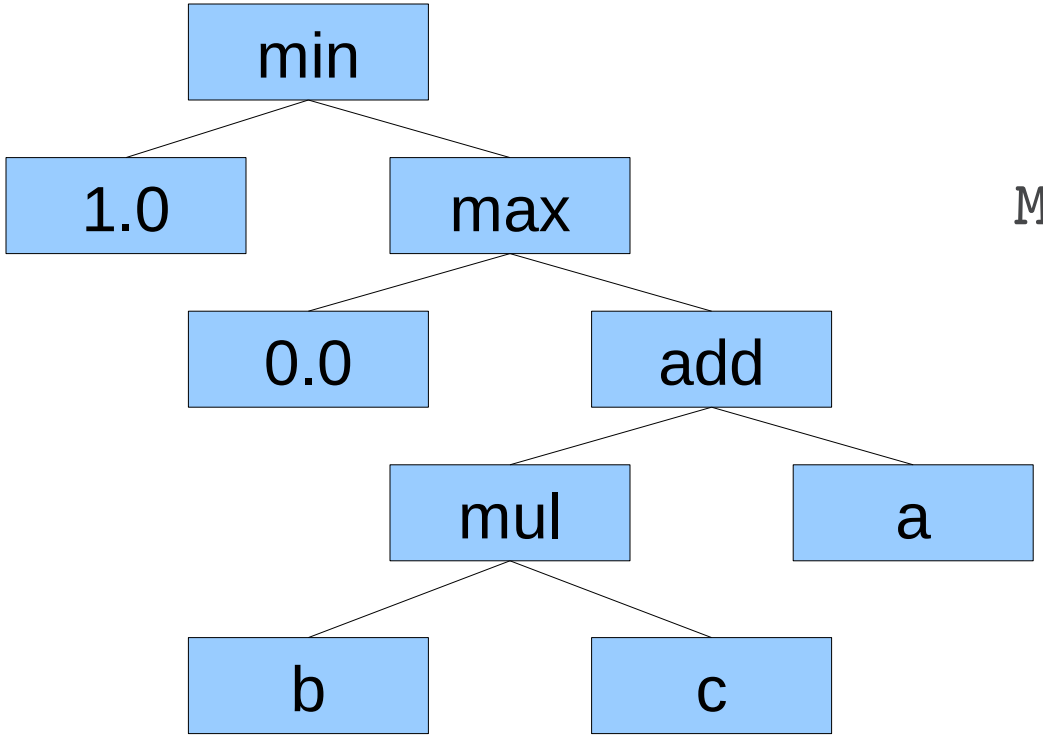


Tree-like IR – Expressions



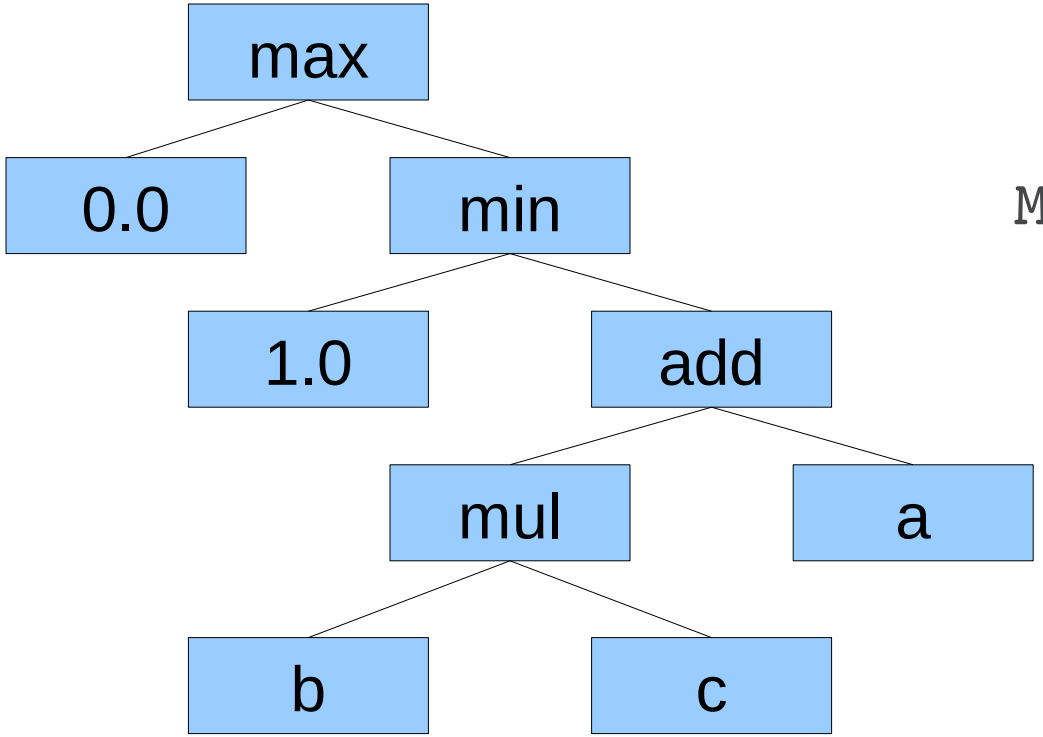

```
MAD_SAT …, b, c, a
```


Tree-like IR – Expressions




```
MAD_SAT …, b, c, a
```


Tree-like IR – Expressions




```
MAD_SAT …, b, c, a
```


Tree-like IR – Expressions

```
void main()
{
    x = a * b;
    …
    y = x + c;
    …
    gl_Position = max(0., min(1. y));
}
```


Tree-like IR – Expressions

```
class ir_rvalue : public ir_instruction {
   const struct glsl_type *type;
};

class ir_expression : public ir_rvalue {
   ir_expression_operation operation;
   ir_rvalue *operands[4];
};

class ir_dereference : public ir_rvalue { };

class ir_dereference_variable : public ir_dereference {
   ir_variable *var;
};
```


Tree-like IR – Expressions

- Currently three kinds of r-value:
  - Expression
  - Variable dereference
  - Constant

Tree-like IR – Assignments

```
class ir_assignment : public ir_instruction {
   ir_dereference *lhs;
   ir_rvalue *rhs;
   ir_rvalue *condition;
   unsigned write_mask:4;
};
```


Memory and registers

- Put all variables in one of two categories:
  - Anything that is or has an array → lay it out in memory like a UBO
  - Anything else → assign it one or more fake registers from an infinite register pool
  - Also include swizzle information on register usage

Memory and registers

```
class ir_register : public ir_rvalue {
    unsigned reg;
    ir_swizzle_mask swizzle;
};
```


Flat-land

```
class ir_calculation : public ir_instruction {
   ir_register lhs;
   ir_expression_operation operation;
   ir_register operands[4];
   unsigned write_mask:4;
};
```


Flat-land

- Connect definitions with uses via “simplified” ud-chains

Flat-land

- Connect definitions with uses via “simplified” ud-chains

Variants of SSA form have been used for detecting program equivalence [5, 52] and for increasing parallelism in imperative programs [21]. The representation of simple data flow information (def-use chains) may be made more compact through SSA form. If a variable has D definitions and U uses, then there can be $D \times U$ def-use chains. When similar information is encoded in SSA form, there can be at most E def-use chains, where E is the number of edges in the control flow graph [40]. Moreover, the def-use information

R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451-490, Oct 1991.

UD-Chains

```
if (condition) {
    x = dot(normal, eye_vector);
} else {
    x = dot(normal, light_vector);
}

x = clamp(x, 0.0, 1.0);
```


Other dirty laundry

- No explicit basic blocks
- No high-level IR for switch-statements
- Explicit AST

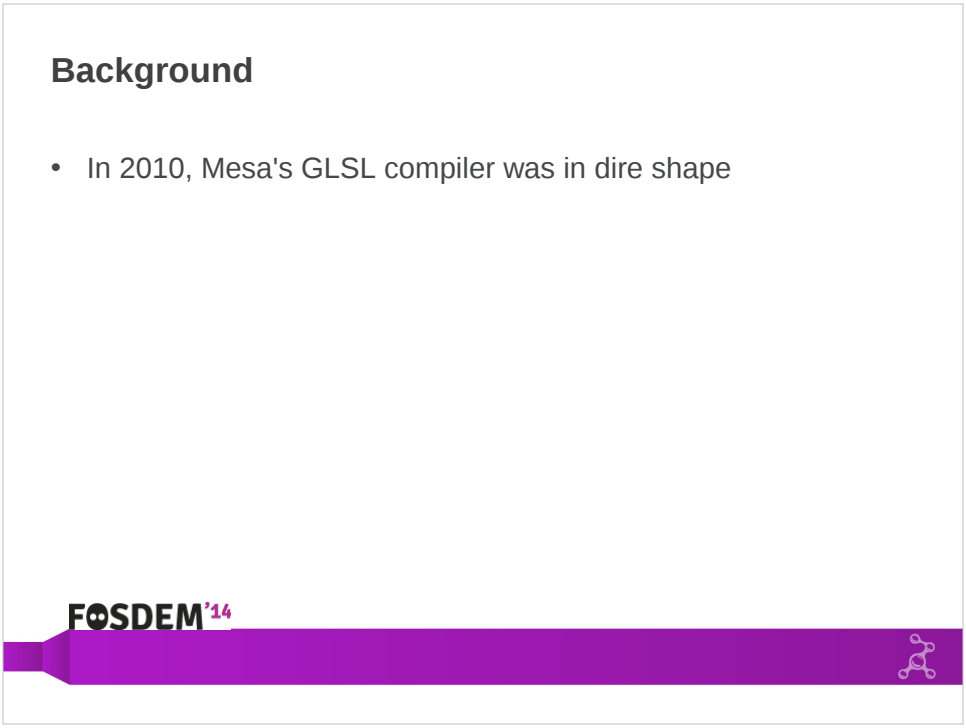

TURE INTEL LINUX WIRELESS GUPNP KVM POKY
OP OFONO LINUX KERNEL
CS YOCTO CONNMAN XEN
SYNCEVOLUTION SIMPLE FIRMWARE INTERFACE (SFI) ENTERPRISE SECURITY INFRASTRUCTURE
INTEL OPEN SOURCE
TECHNOLOGY CENTER

Three years experience with a tree-like shader IR

Ian Romanick
ian.d.romanick@intel.com
1-February-2014

Agenda

- Background
- Problems with current IR
- Steps towards a better IR



Technically supported GLSL 1.20, but ma ny shaders failed to compile.

Written using a custom parser generator... only one person in the world had any idea how it work

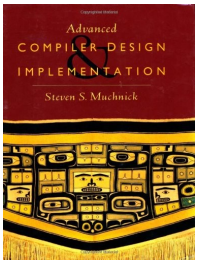
It used a stack-like register set... 80387 anybody?

No hope of getting GLSL 1.30 or reasonable optimizations done in any desirable timeframe.

In fairness Michal Krol wrote it as an undergrad thesis project, and it was pretty amazing in that light.

Background

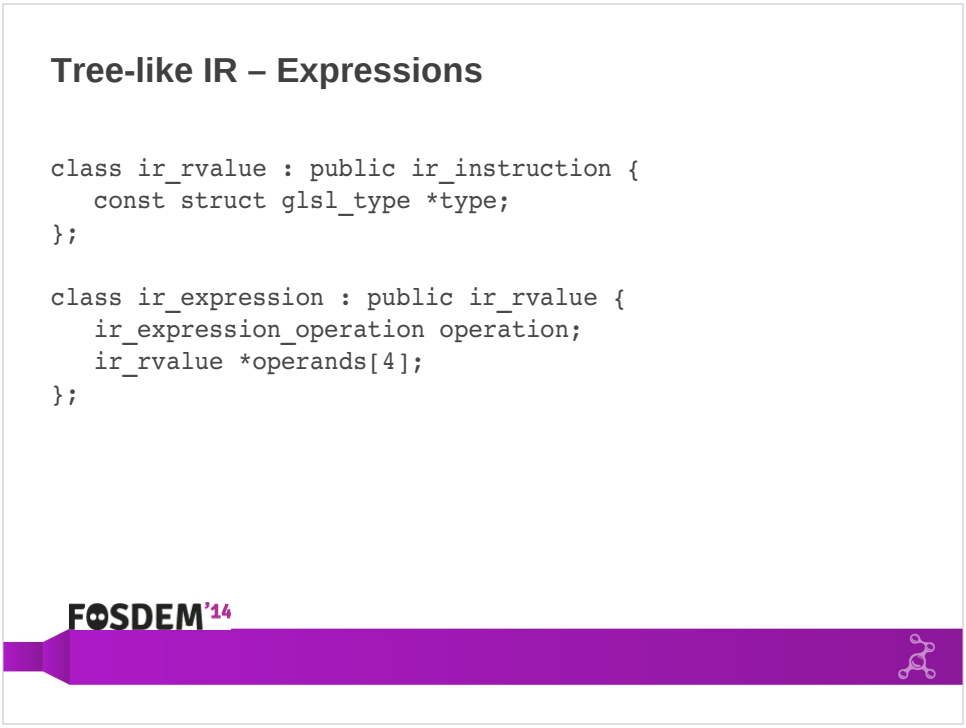
- Complete compiler re-write
  - “Advanced Compiler Design and Implementation” by Steven Muchnick
  - “A Retargetable C Compiler: Design and Implementation” by David Hanson



Muchnick book published just before SSA became mainstream.

Hanson uses a separate machine description language to generate least-cost code generators using a fairly sophisticated dynamic programming technique.... this wants to have as much information about how intermediate values are consumed as possible.

It also wants to be integrated with CSE and register allocation.



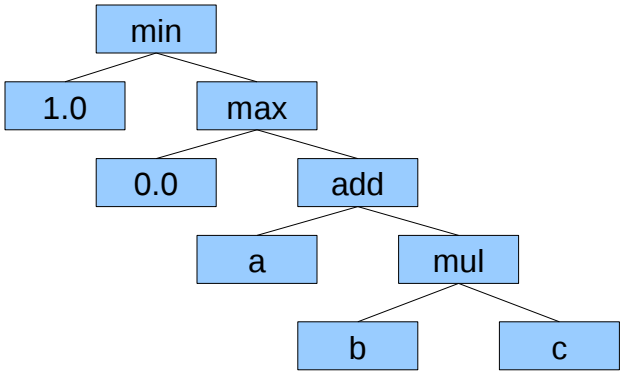
We had three visions for this:

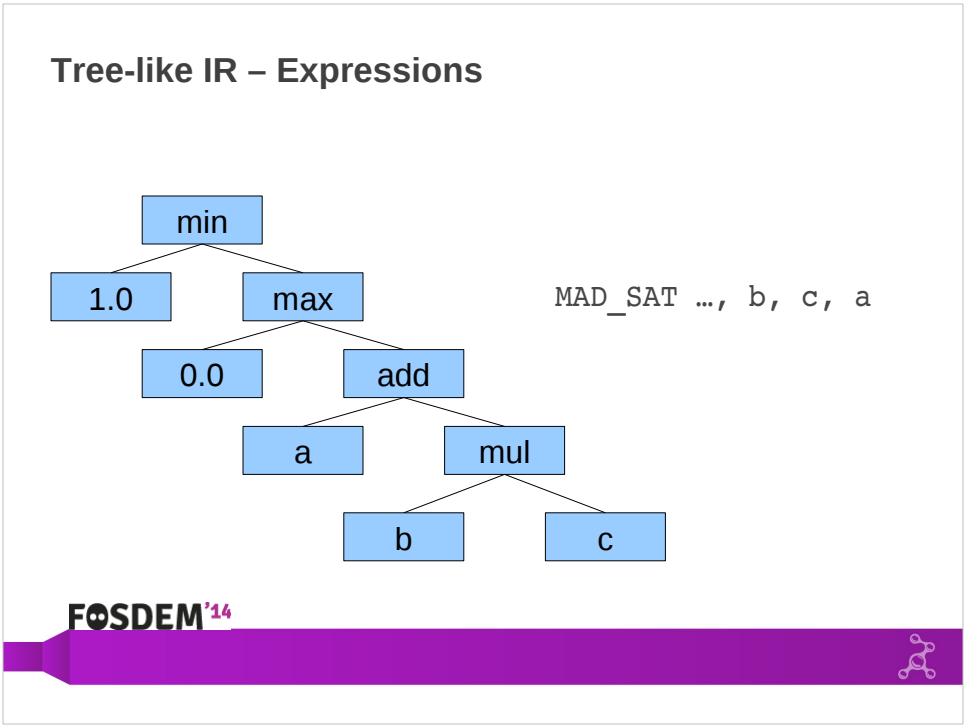
1. Be able to create a machine description language to ease supporting new hardware (new chips in the same family or new families).

2. Be able to easily recognize complex patterns to generate optimal instruction sequences.

3. Be able to easily drop in new optimization passes that would operate independently of other passes.

Tree-like IR – Expressions





The compiler detects several sequences like this:

max(0, min(1, ...) → saturate

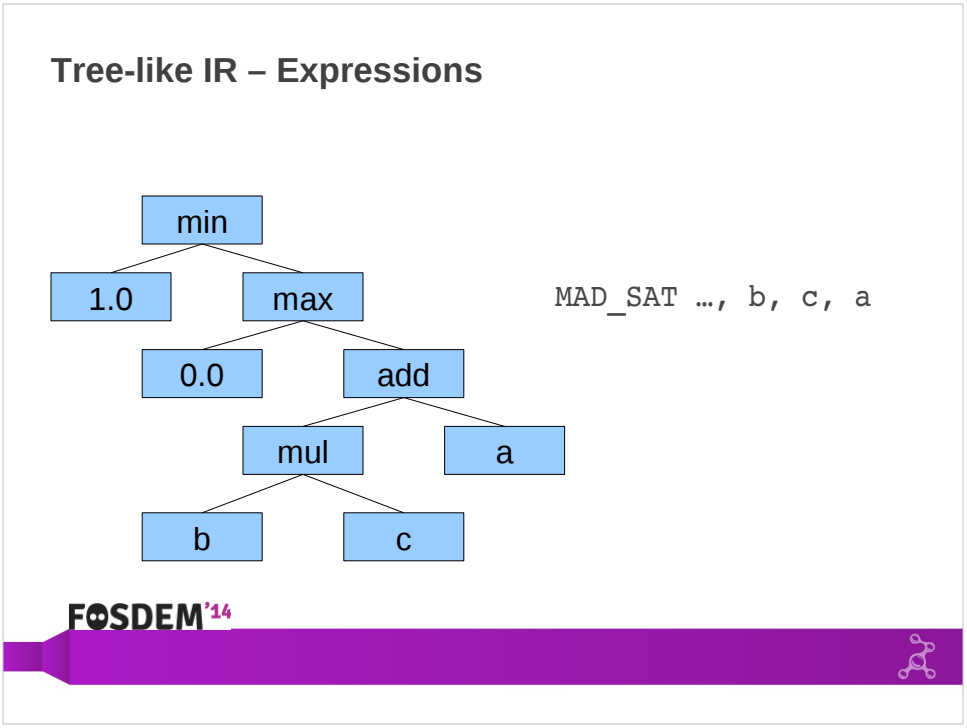
add(mul(..., ...), ...) → MAD

various combinations → LRP

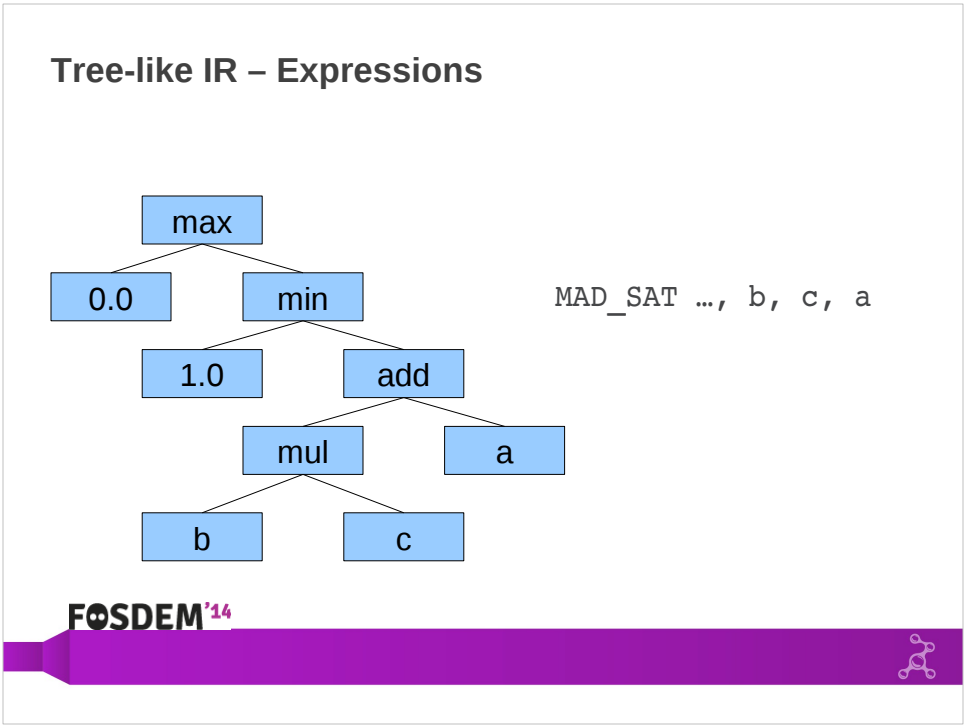
Algebraic optimizations operate similarly

pow(2., x) → ex2(x)

GPU instruction sets are generally very regular and there is no complex addressing (in shaders or in the GPU instruction sets), so we don't actually need most of this potential.



And all of the other possible rotations of the tree.



What about tree sequences that result in multiple (or many!) instructions?

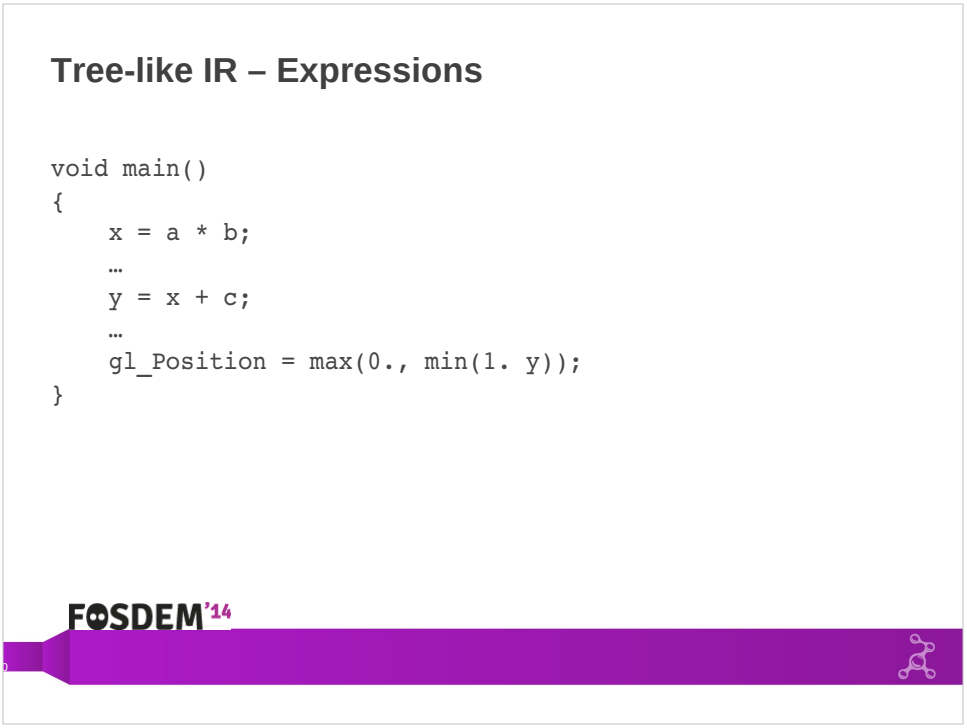
- The code generator has to manage generating and tracking intermediate values. This is a hassle.

What about CSE?

- Forget about it.

And there might be swizzles in there too.

- I don't think the code currently handles it very well if there's a swizzle between the add and the multiple



Instead of a single tree with all the subexpressions, you have a forest of trees to generate the final result.

A transformation pass called “tree grafting” tries to combine disjoint trees into a single tree.

Tree-like IR – Expressions

```
class ir_rvalue : public ir_instruction {
   const struct glsl_type *type;
};

class ir_expression : public ir_rvalue {
   ir_expression_operation operation;
   ir_rvalue *operands[4];
};

class ir_dereference : public ir_rvalue { };

class ir_dereference_variable : public ir_dereference {
   ir_variable *var;
};
```

FOSDEM'14

More complex dereferences (e.g., arrays or structures) involve more nodes in the tree.

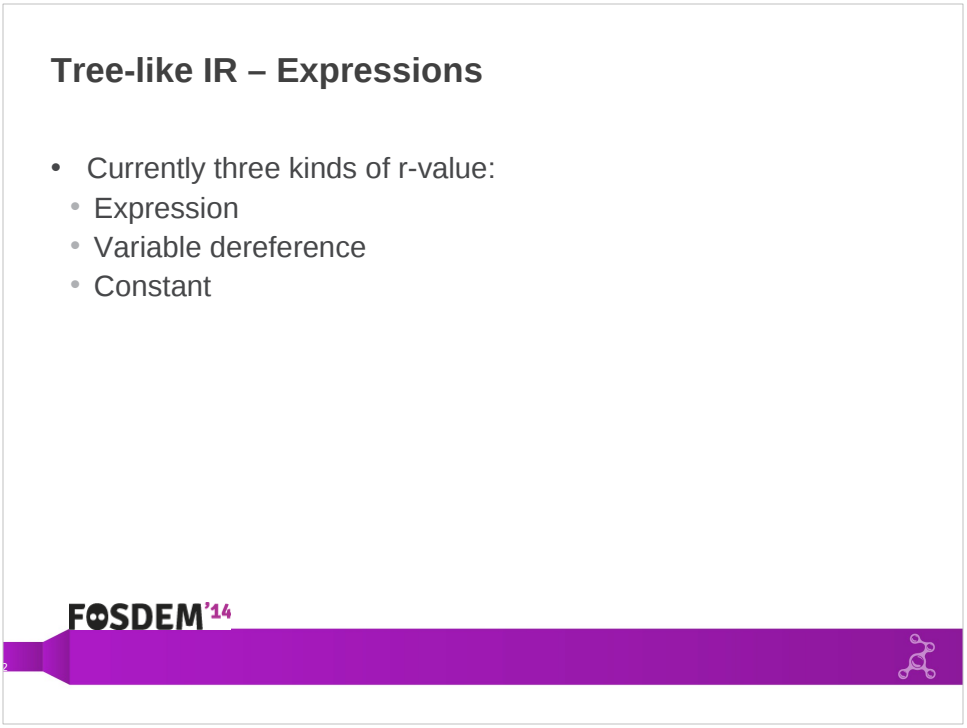
Lots of pointers → huge memory usage

Some games have trouble compiling all of their shaders on 32-bit machines due to the memory usage.

Walking trees is not very cache friendly, and we have many independent optimization passes.

Code generation is difficult because instructions are generated bottom-up. The recursive process has to pass instructions back up the tree so that they can be modified.

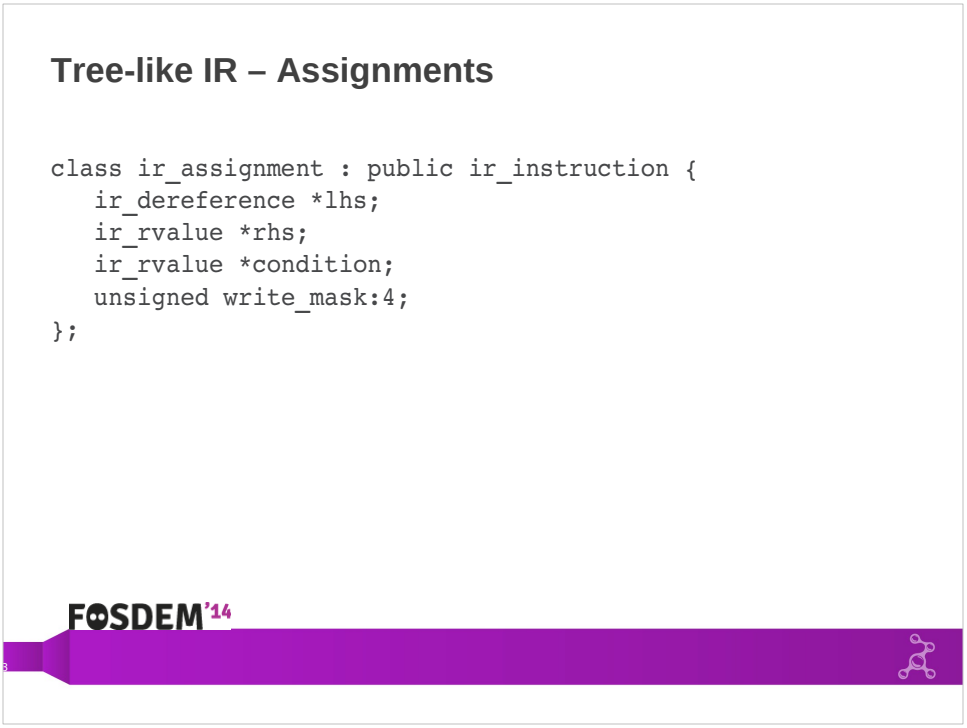
> In the previous example, the add would notice the multiply “below” it, and generate the MAD. This would get passed back up to the min/max nodes to generate the _SAT modifier.



Each of these is a subclass of ir_rvalue.

ir_dereference can be an ir_dereference_variable or array / structure dereference.

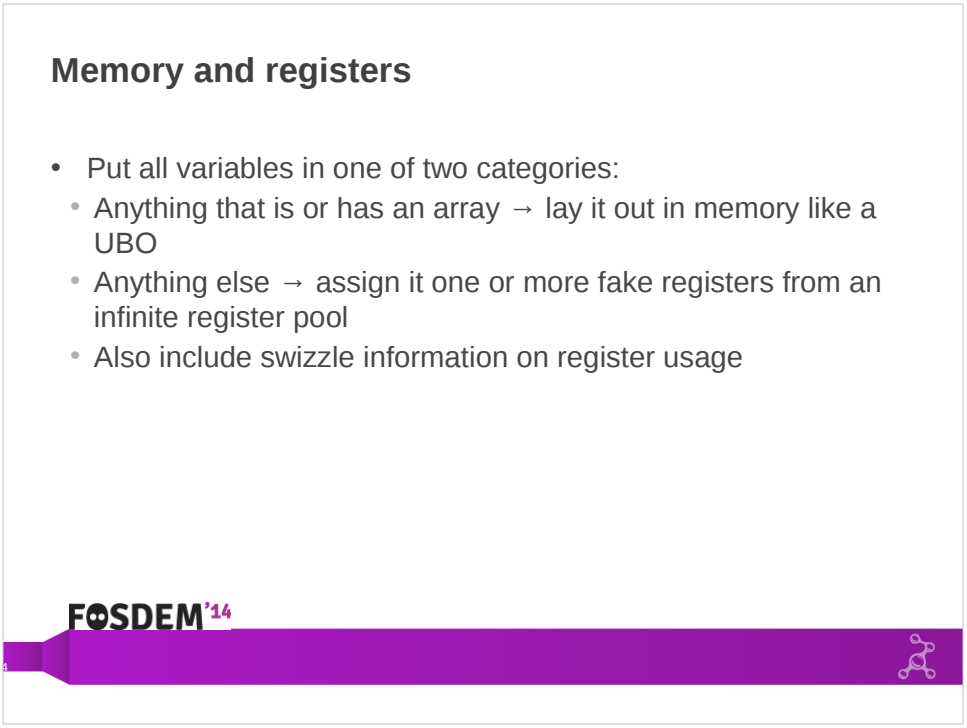
ir_dereference_variable is used for temporaries, uniforms, attributes, varyings, and fragment outputs.



The assignment node has pretty much the expected elements.

One odd bit is the condition field

- We intended this to allow every assignment to possibly be a conditional assignment.
- Since the RHS is potentially a giant, complex tree, this has dubious merit in practice.



Keep a mapping from fake register number to the ir_variable

The mapping requires storage, but it is much smaller than the ir_dereference system for 3 reasons:

1. One mapping per variable instead of per access of the variable.
2. Generated temporaries don't need a mapping or an ir_variable.
3. After linking, the mapping can be freed.

Keep a mapping from a base address to the ir_variable

Arrays are always accessed by loads / stores that cannot appear in trees.

Perform two related optimization passes:

Kill redundant loads (and stores) from arrays with constant indices

If all of the accesses have constant indices, kill the mapping

In conjunction with the existing lowering passes, this should keep parity with today

It's also worth noting that we could treat ir_constant the same way

Registers can be generated directly from the AST, and the whole ir_dereference class hierarchy can die.

Memory and registers

```
class ir_register : public ir_rvalue {
    unsigned reg;
    ir_swizzle_mask swizzle;
};
```

This still includes type information from ir_rvalue.

On LP64 systems, this is the same size as ir_dereference_variable, but it is larger on LP32.

The advantage is you only need one of them, and you no longer need ir_swizzle either.

We're reducing the depth of the trees.

You can almost smell the transition to SSA coming...

There are other ir_rvalues and related things that need to be handled

- ir_call – Not really an ir_rvalue, but it comes into play
- ir_texture – We'll likely want to change these to behave more like the new ir_calculation instructions... probably extend ir_call for “intrinsic” operations. Existing texture operations are pushing the limits of fitting everything in a fixed set of operands.
- All three might even end up with a new, shared base class.

But we've lost some information here, and we use that information for various optimization passes and, as previously noted, for some code generation.

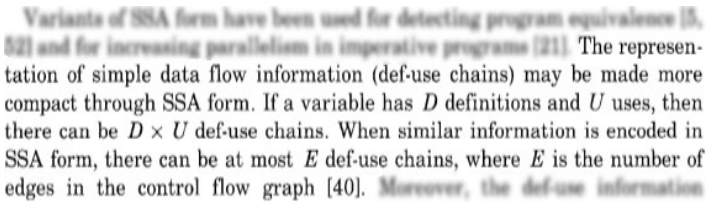
- We no longer have any information about how values are consumed.
- Even if we generate MAD_SAT directly while creating ir_calculation, later optimizations passes may make new opportunities available. This is especially true for LRP.
- We could go back to ir_expression, do tree grafting, come back to ir_calculation, etc... YUCK!

Flat-land

- Connect definitions with uses via “simplified” ud-chains

Flat-land

- Connect definitions with uses via “simplified” ud-chains



Variants of SSA form have been used for detecting program equivalence [5, 52] and for increasing parallelism in imperative programs [21]. The representation of simple data flow information (def-use chains) may be made more compact through SSA form. If a variable has D definitions and U uses, then there can be $D \times U$ def-use chains. When similar information is encoded in SSA form, there can be at most E def-use chains, where E is the number of edges in the control flow graph [40]. Moreover, the def-use information

R. Cytron, J. Ferrante, B. Rosen, M. Wegman, and K. Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, 13(4):451-490, Oct 1991.

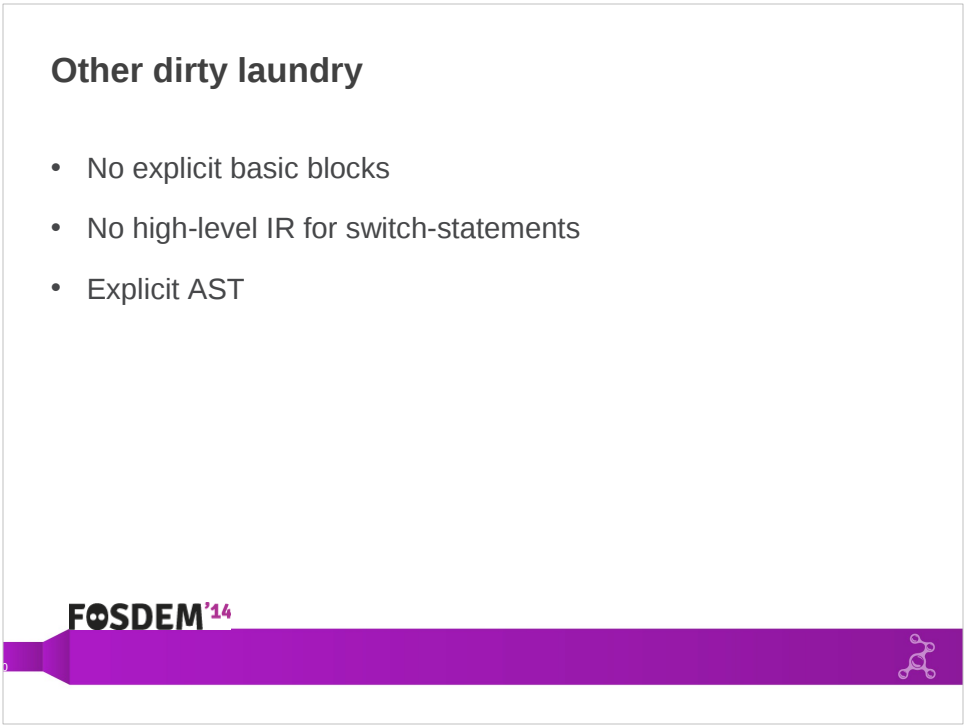
SSA doesn't eliminate the need for UD-chains

- It reduces the need, and it provides a number of simplifying assumptions that aid construction.

For “simplified” ud-chains we really need to know cases when the linkage is strictly forward within a basic block

- This can be generated directly in the transition from tree to flat.
- It can also be updated using very similar code to the tree-grafting pass.

With or without SSA, UD-chains help us move the clamp inside the branches to generate DP3_SAT.



With the transition to flat-land, we also want to track basic blocks explicitly. Without trees, keeping BBs up to date when code changes should be easier. This was previously very annoying with trees.

We also lack an explicit IR for switch-statements. Instead complex sequences of if-statements are generated. On many GPUs better code can be generated if the switch variable is uniformly constant. This hasn't been high-priority yet due to so few (i.e., zero) applications using switch-statements.

When we started the new compiler, all generated files were tracked in source control. Due to the pain caused by this, we wanted to make as few changes to glsl_parser.y as possible. Since we don't do that, we should get rid of the explicit AST generation.


TURE INTEL LINUX WIRELESS GUPNP KVM POKY
DE YOCTO CONNMAN XEN OFONO LINUX KERNEL
CS SYNCEVOLUTION SIMPLE FIRMWARE INTERFACE (SFI) ENTERPRISE SECURITY INFRASTRUCTURE
INTEL OPEN SOURCE
TECHNOLOGY CENTER