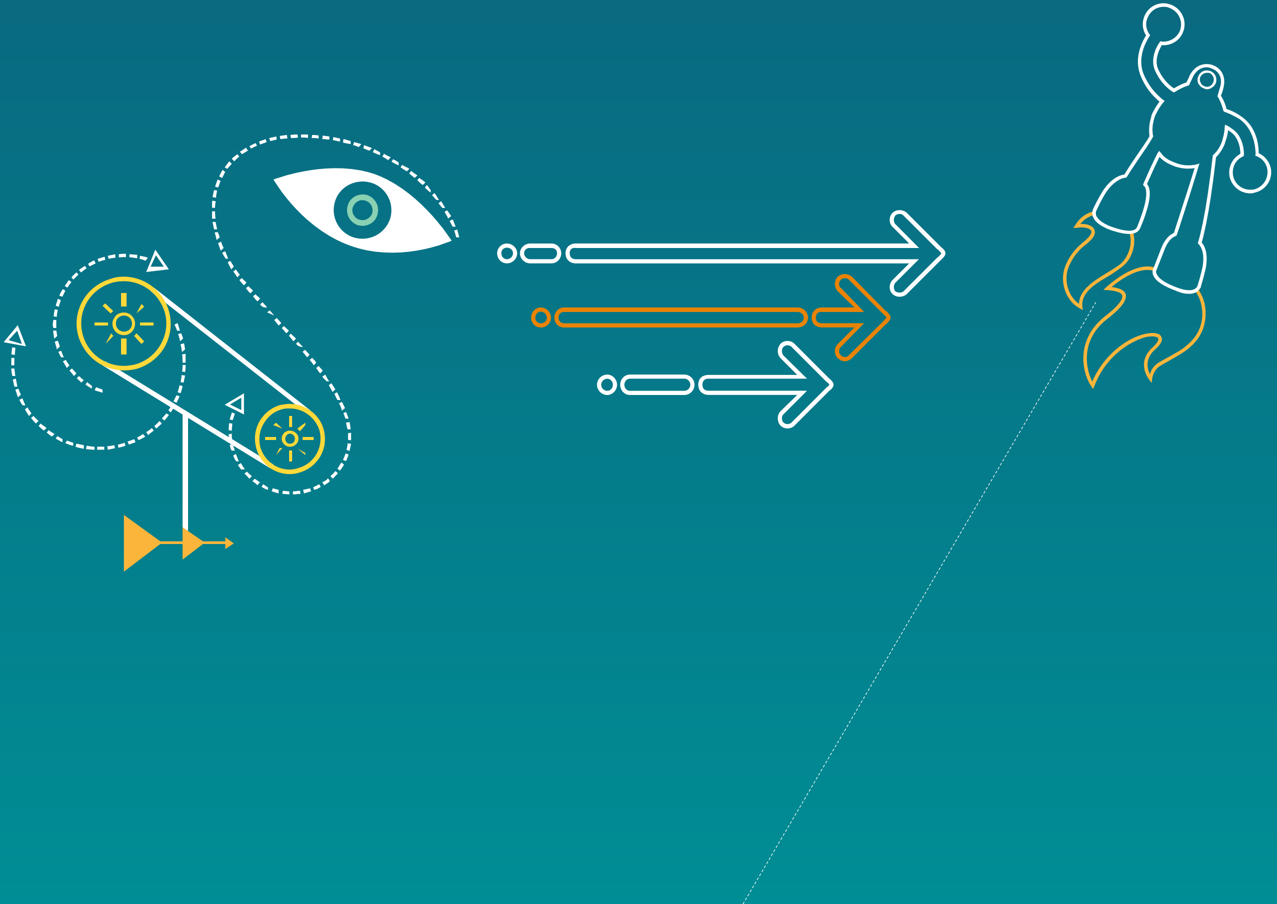
Dave Estes - Senior Staff Engineer
Qualcomm Innovation Center, Inc.

SchedMachineModel: Adding and Optimizing a Subtarget

QUALCOMM®



Demo Code at:

https://www.codeaurora.org/patches/quic/llvm/77947/

1. Scheduling Overview
2. MIScheduler
3. SchedMachineModel
4. Basic Model Example
5. Refined Model Example

Agenda

Scheduling Overview

- Static Instruction Scheduling (Compile Time)
  - Ordering of instruction stream to minimize stalls and increase IPC
  - Critical for VLIW, still really important for simple in-order and out-of-order superscaler machines
- Dynamic Instruction Scheduling (On Device)
  - Selectively issuing instructions out-of-order to minimize stalls and increase IPC

LLVM Schedulers

- Pre 2008: SelectionDAGISel pass creates the ScheduleDAG from the SelectionDAG at the end of instruction selection
- ScheduleDAG works on SelectionDAG Nodes (SDNodes)

```
// Scheduler Class Hierarchy

ScheduleDAG
• ScheduleDAGFast
• ScheduleDAGRRList
```

LLVM Schedulers

- Circa 2008: Post Register Allocation pass added for instruction selection
- SchedulePostRATDList works on MachineInstrs

```
// Scheduler Class Hierarchy

ScheduleDAG
• ScheduleDAGSDNodes
    • ScheduleDAGFast
    • ScheduleDAGRRList
• ScheduleDAGInstrs
    • SchedulePostRATDList
```

LLVM Schedulers

- Circa 2012: MIScheduler (ScheduleDAGMI) added as separate pass for pre-RA scheduling
- Circa 2014: MIScheduler adapted to optionally replace PostRA Scheduler

```
// Scheduler Class Hierarchy

ScheduleDAG
• ScheduleDAGSDNodes
    • ScheduleDAGFast
    • ScheduleDAGRRList
    • ScheduleDAGLinearize
    • ScheduleDAGVLIW
• ScheduleDAGInstrs
    • DefaultVLIWScheduler
    • ScheduleDAGMI
        • ScheduleDAGMILive
            • VLIWMachineScheduler
    • SchedulePostRATDList
```

Agenda

MIScheduler

- MIScheduler is slowly being adapted as the scheduler of the future
- AArch64 backend uses MIScheduler exclusively
- List Scheduler suitable for VLIW, out-of-order, and in-order machines
- Schemes: Top-Down, Bottom-Up, or Bi-Directional
- Heuristics: Register Pressure, Latency, Clustering, Critical Resource

Using MIScheduler

- Enabled with -enable-misched and -misched-postra
- Optionally can override your target’s TargetSubtargetInfo methods enableMachineScheduler() and enablePostMachineScheduler().
- Force scheme with -misched-topdown or -misched-bottomup
- Enable additional analysis / heuristics with -misched-cluster, -misched-cyclicpath, -misched-regpressure, and -misched-fusion
- Set scheduler (strategy) with -misched=(default, converge, ilpmax, ilpmin, or shuffle)

Extending MIScheduler

- The pass calls MachineSchedulerBase::scheduleRegions() for each machine function
- scheduleRegions() calls ScheduleDAG::schedule() on each region
- schedule() uses the MachineSchedStrategy implementation to choose candidate instruction
- Customization Options (see MachineScheduler.h):
  - Create entire new pass
  - Override DAG builder and scheduler
  - Create an alternative MachineSchedStrategy

```
// The pass
MachineFunctionPass
• MachineSchedulerBase
    • MachineScheduler

// The scheduler
ScheduleDag
• ScheduleDAGInstrs
    • ScheduleDAGMI
        • ScheduleDAGMILive

// The strategy
MachineSchedStrategy
• ILPScheduler
• InstructionShuffler
• ConvergingVLIWScheduler
• GenericSchedulerBase
    • GenericScheduler
    • PostGenericScheduler
• R600SchedStrategy
```

1 Scheduling Overview

2 MIScheduler

3 SchedMachineModel

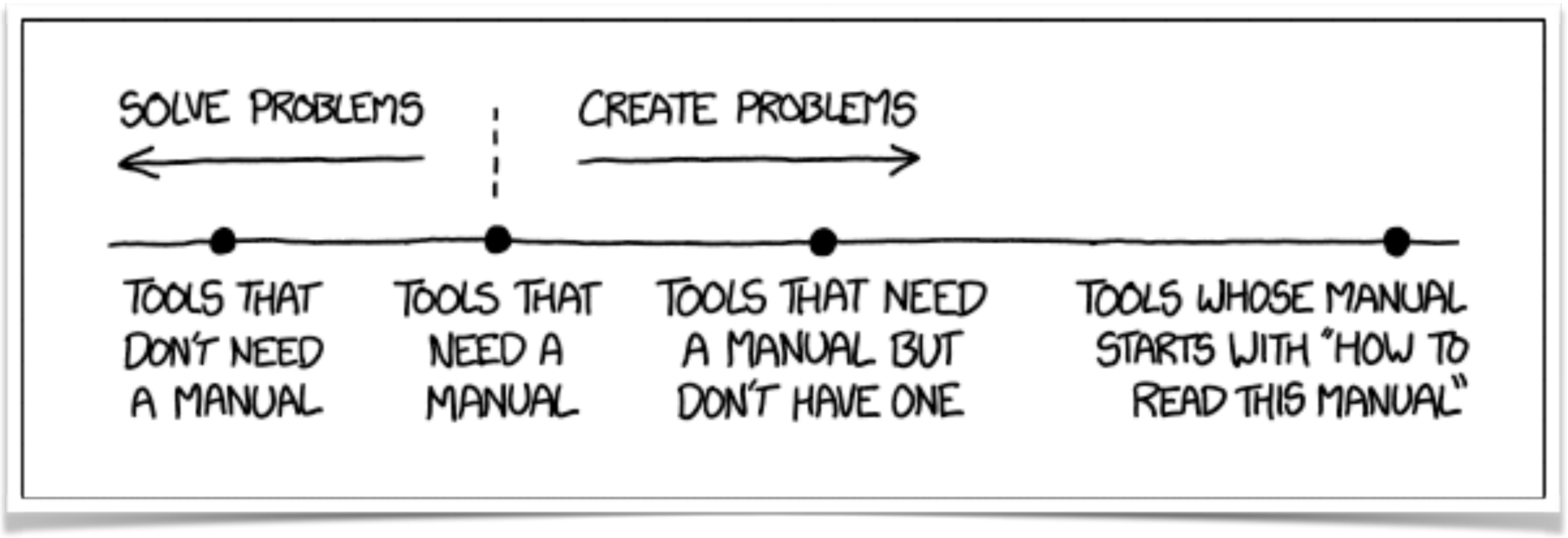
4 Basic Model Example

5 Refined Model Example

Agenda

The Fun Part: TableGen

- SchedMachineModel is defined with TableGen
- RTM: http://llvm.org/docs/TableGen/index.html


SOLVE PROBLEMS
CREATE PROBLEMS
TOOLS THAT DON'T NEED A MANUAL
TOOLS THAT NEED A MANUAL
TOOLS THAT NEED A MANUAL BUT DON'T HAVE ONE
TOOLS WHOSE MANUAL STARTS WITH "HOW TO READ THIS MANUAL"

Using TableGen

- Key Target and Subtarget details are defined with a TableGen Definition (.td) file
- TableGen Generators

  --gen-register-info

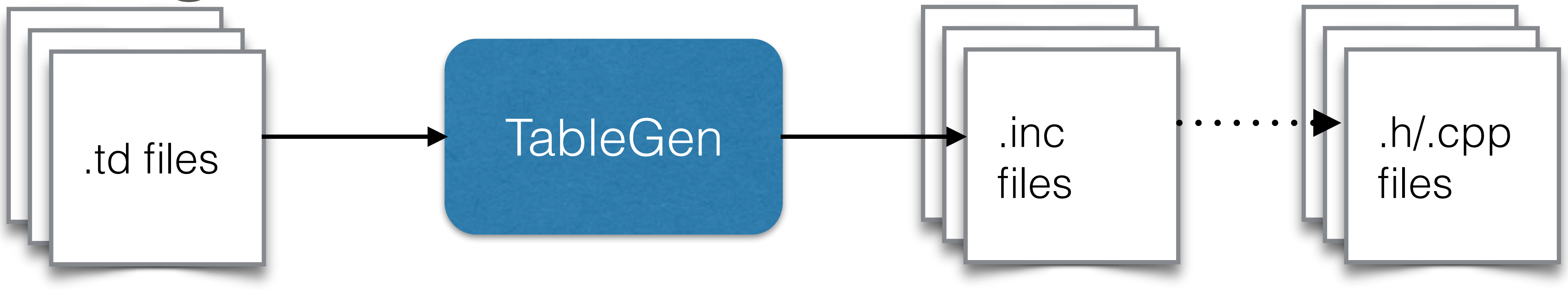
  --gen-instr-info

  --gen-subtarget

  --print-records

```
$ cd llvm/lib/Target/AArch64
$ ls *.td -c1
AArch64RegisterInfo.td
AArch64SchedA53.td
AArch64SchedA57.td
AArch64SchedA57WriteRes.td
AArch64SchedCyclone.td
AArch64Schedule.td
AArch64InstrFormats.td
AArch64InstrInfo.td
AArch64CallingConvention.td
AArch64InstrAtomics.td
AArch64.td
```

Including TableGen'd Data



AArch64SchedA53.td

```
def CortexA53Model : SchedMachineModel {
  let MicroOpBufferSize = 0;
  let IssueWidth = 2;
  let MinLatency = 1;
  let LoadLatency = 3;
  let MispredictPenalty = 9;
}
```

AArch64MCTargetDesc.cpp

```
#define GET_SUBTARGETINFO_MC_DESC
#include "AArch64GenSubtargetInfo.inc"
```

AArch64GenSubtargetInfo.inc

```
static const llvm::MCSchedModel CortexA53Model = {
  2, // IssueWidth
  0, // MicroOpBufferSize
  MCSchedModel::DefaultLoopMicroOpBufferSize,
  3, // LoadLatency
  MCSchedModel::DefaultHighLatency,
  9, // MispredictPenalty
  0, // PostRAScheduler
  1, // CompleteModel
  1, // Processor ID
  CortexA53ModelProcResources,
  CortexA53ModelSchedClasses,
  8,
  452,
  nullptr}; // No Itinerary
```

TableGen Basics

- Records: a name, list of values, and list of superclasses
  - `def`: concrete form of records
  - `class`: abstract form of records
  - `multiclass`: groups of abstract records
- Rich primitive types, loops, conditionals, arithmetic operators, and lists.

SchedMachineModel Structure

- llvm/include/llvm/Target/TargetSchedule.td
- llvm/include/MC/MCSchedule.h

```
class SchedMachineModel {
  int IssueWidth = -1; // Max micro-ops that may be scheduled per cycle.
  int MinLatency = -1; // Determines which instructions are allowed in a group.
                       // (-1) inorder (0) ooo, (1): inorder +var latencies.
  int MicroOpBufferSize = -1; // Max micro-ops that can be buffered.
  int LoopMicroOpBufferSize = -1; // Max micro-ops that can be buffered for
                                  // optimized loop dispatch/execution.
  int LoadLatency = -1; // Cycles for loads to access the cache.
  int HighLatency = -1; // Approximation of cycles for "high latency" ops.
  int MispredictPenalty = -1; // Extra cycles for a mispredicted branch.

  // Per-cycle resources tables.
  ProcessorItineraries Itineraries = NoItineraries;

  bit PostRAScheduler = 0; // Enable Post RegAlloc Scheduler pass.
```

SchedMachineModel

- Cortex-A53 Sample
- Each Subtarget should define a SchedMachineModel

```
// Cortex-A53 machine model for scheduling and other instruction cost
heuristics.
def CortexA53Model : SchedMachineModel {
  let MicroOpBufferSize = 0; // Explicitly set to zero since A53 is in-order.
  let IssueWidth = 2;        // 2 micro-ops are dispatched per cycle.
  let MinLatency = 1 ;       // OperandCycles are interpreted as MinLatency.
  let LoadLatency = 3;       // Optimistic load latency assuming bypass.
                             // This is overriden by OperandCycles if the
                             // Itineraries are queried instead.
  let MispredictPenalty = 9; // Based on microarchitecture software
                             // optimization guidelines
}
```

ProcResourceUnits

- Define the processor’s resources which impact scheduling
- Pipelines, functional units, issue ports, etc.

```
// Modeling each pipeline as a ProcResource using the BufferSize = 0 since
// Cortex-A53 is in-order.

def A53UnitALU    : ProcResource<2> { let BufferSize = 0; } // Int ALU
def A53UnitMAC    : ProcResource<1> { let BufferSize = 0; } // Int MAC
def A53UnitDiv    : ProcResource<1> { let BufferSize = 0; } // Int Division
def A53UnitLdSt   : ProcResource<1> { let BufferSize = 0; } // Load/Store
def A53UnitB      : ProcResource<1> { let BufferSize = 0; } // Branch
def A53UnitFPALU  : ProcResource<1> { let BufferSize = 0; } // FP ALU
def A53UnitFPMDS  : ProcResource<1> { let BufferSize = 0; } // FP Mult/Div/Sqrt
```

SchedReadWrite

- SchedReadWrite
  - SchedWrite: output operand schedule information
  - SchedRead: input operand schedule information
- Each instruction’s output operand(s) is annotated with a default target SchedWrite
- Some instructions’ input operands are annotated with a default target SchedRead

WriteRes

- Defines new subtarget SchedWriteRes that maps resources the for a target SchedWrite
- Specifies which resources are required, duration, whether pipelined, and hazards

```
let SchedModel = CortexA53Model in {

// ALU - Despite having a full latency of 4, most of the ALU instructions can
//       forward a cycle earlier and then two cycles earlier in the case of a
//       shift-only instruction. These latencies will be incorrect when the
//       result cannot be forwarded, but modeling isn't rocket surgery.
def : WriteRes<WriteImm, [A53UnitALU]> { let Latency = 3; }
def : WriteRes<WriteI, [A53UnitALU]> { let Latency = 3; }
def : WriteRes<WriteISReg, [A53UnitALU]> { let Latency = 3; }
def : WriteRes<WriteIEReg, [A53UnitALU]> { let Latency = 3; }
def : WriteRes<WriteIS, [A53UnitALU]> { let Latency = 2; }
def : WriteRes<WriteExtr, [A53UnitALU]> { let Latency = 3; }
```

ReadAdvance

- Defines new subtarget SchedReadAdvance that maps forwarding information for a target SchedRead
- Used to model forwarding
- Considered an “advanced” modeling feature

```
// No forwarding for these reads.
def : ReadAdvance<ReadI, 0>;
def : ReadAdvance<ReadIM, 0>;
def : ReadAdvance<ReadIMA, 0>;
def : ReadAdvance<ReadExtrHi, 0>;
def : ReadAdvance<ReadAdrBase, 0>;
def : ReadAdvance<ReadVLD, 0>;
```

Modeling Strategy

- Create Basic Model
  - Define SchedMachineModel
  - Define processor resources
  - Map processor resources to default target SchedWrites
- Refine Basic Model
  - Improve instruction scheduling information
  - Add forwarding
  - Add hazards
  - Optionally model key features of micro-architecture

1 Scheduling Overview

2 MIScheduler

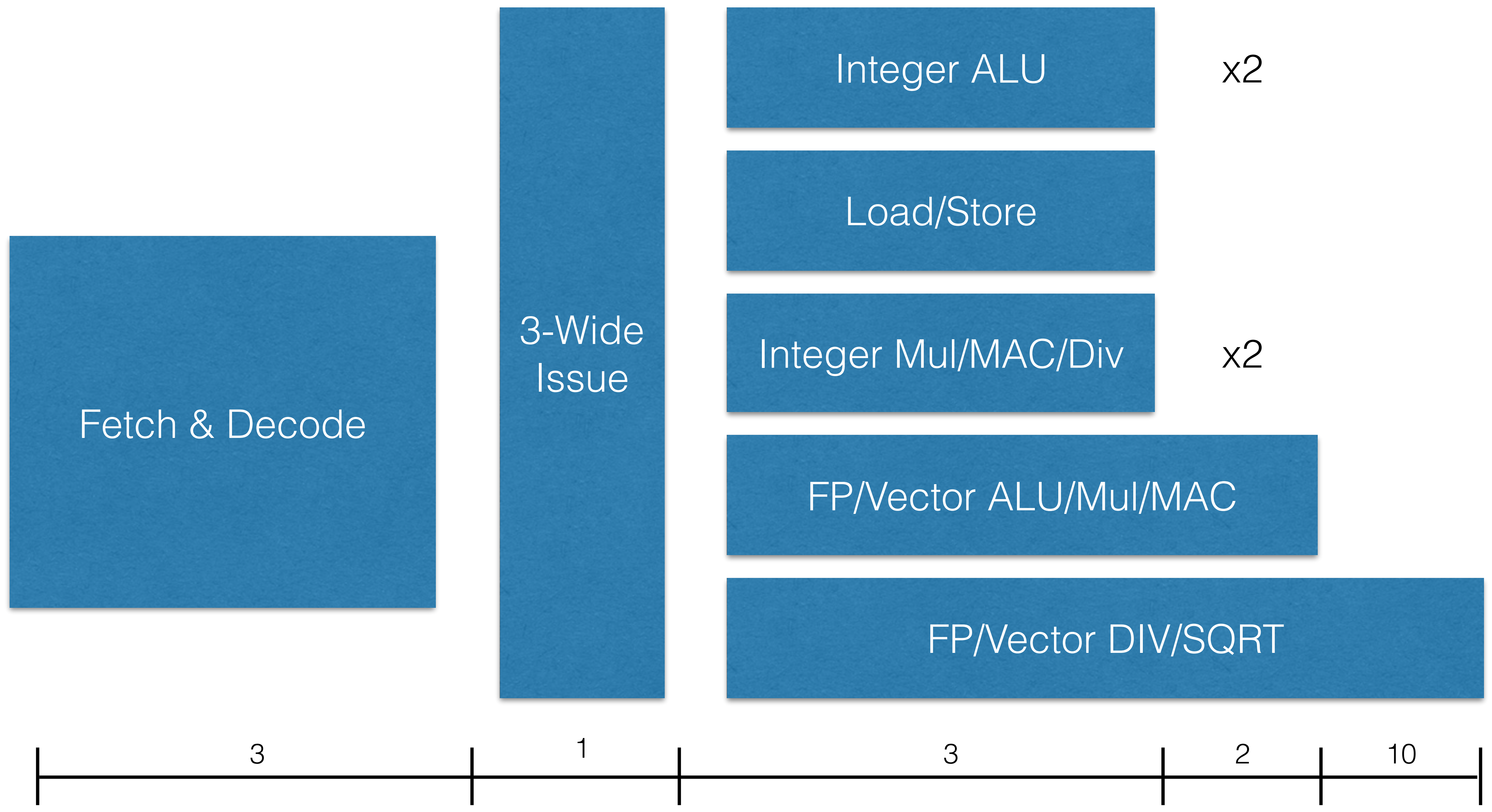
3 SchedMachineModel

4 Basic Model Example

5 Refined Model Example

Agenda

Simple In-Order Machine



Demonstrate: Implement

1. Edit AArch64.td to add new subtarget
2. Create AArch64SchedDemo.td
3. Add SchedMachineModel
4. Add ProcResources
5. Create each SchedWriteRes
6. Create each SchedReadAdvance and zero
7. Build

Demo Code at:
https://www.codeaurora.org/patches/quic/llvm/77947/

Demonstrate: Evaluate

1. Compile a test with debug output
2. Go over the output observing candidate reasons
3. Illustrate example lit test

1 Scheduling Overview

2 MIScheduler

3 SchedMachineModel

4 Basic Model Example

5 Refined Model Example

Agenda

InstRW

- InstRW is used to refine instruction scheduling information for the subtarget, overriding the target defaults

```
// Miscellaneous
def : InstRW<[WriteI], (instrs COPY)>;

// Defining new, named SchedWrites for re-use within the subtarget
def A53WriteVLD1 : SchedWriteRes<[A53UnitLdSt]> { let Latency = 4; }
def A53WriteVLD2 : SchedWriteRes<[A53UnitLdSt]> { let Latency = 5;
                                                  let ResourceCycles = [2]; }

// Using the new SchedWrites to instructions matched by regex
def : InstRW<[A53WriteVLD1], (instregex "LD1Onev(8b|4h|2s|1d|16b|8h|4s|2d)$")>;
def : InstRW<[A53WriteVLD2], (instregex "LD1Twov(8b|4h|2s|1d|16b|8h|4s|2d)$")>;

def : InstRW<[A53WriteVLD1, WriteAdr],
             (instregex "LD1Rv(8b|4h|2s|1d|16b|8h|4s|2d)_POST$")>;
def : InstRW<[A53WriteVLD2, WriteAdr],
             (instregex "LD1Twov(8b|4h|2s|1d|16b|8h|4s|2d)_POST$")>;
```

ReadAdvance

- Defines new subtarget SchedReadAdvance that maps forwarding information for a target SchedRead

```
// ALU - Most operands in the ALU pipes are not needed for two cycles.
def : ReadAdvance<ReadI, 2, [WriteImm,WriteI,
                             WriteISReg, WriteIEReg,WriteIS,
                             WriteID32,WriteID64,
                             WriteIM32,WriteIM64]>;

// MAC - Operands are generally needed one cycle later in the MAC pipe.
//       Accumulator operands are needed two cycles later.
def : ReadAdvance<ReadIM, 1, [WriteImm,WriteI,
                              WriteISReg, WriteIEReg,WriteIS,
                              WriteID32,WriteID64,
                              WriteIM32,WriteIM64]>;
def : ReadAdvance<ReadIMA, 2, [WriteImm,WriteI,
                               WriteISReg, WriteIEReg,WriteIS,
                               WriteID32,WriteID64,
                               WriteIM32,WriteIM64]>;
```

SchedVariant

- Used when the scheduling information is variant
- Determined at compile time based on the supplied SchedPredicate

```
// Predicate for determining when a shiftable register is shifted.
def RegShiftedPred : SchedPredicate<[{TII->hasShiftedReg(MI)}]>;


def A53ReadShifted : SchedReadAdvance<1, [WriteImm,WriteI,
                                          WriteISReg, WriteIEReg,WriteIS,
                                          WriteID32,WriteID64,
                                          WriteIM32,WriteIM64]>;
def A53ReadNotShifted : SchedReadAdvance<2, [WriteImm,WriteI,
                                             WriteISReg, WriteIEReg,WriteIS,
                                             WriteID32,WriteID64,
                                             WriteIM32,WriteIM64]>;
def A53ReadISReg : SchedReadVariant<[
   SchedVar<RegShiftedPred, [A53ReadShifted]>,
   SchedVar<NoSchedPred, [A53ReadNotShifted]>]>;
def : SchedAlias<ReadISReg, A53ReadISReg>;
```

WriteSequence

- Used to defined a dependent sequence of SchedWrites
- Latencies are additive
- Cyclone Sample

```
// SCVT/UCVT S/D, Rd = VLD5+V4: 9 cycles.
def CyWriteCvtToFPR : WriteSequence<[WriteVLD, CyWriteV4]>;
def : InstRW<[CyWriteCopyToFPR], (instregex "FCVT[AMNPZ][SU][SU][WX][SD]r")>;

// FCVT Rd, S/D = V6+LD4: 10 cycles
def CyWriteCvtToGPR : WriteSequence<[CyWriteV6, WriteLD]>;
def : InstRW<[CyWriteCvtToGPR], (instregex "[SU]CVTF[SU][WX][SD]r")>;
```

Closing

- Thanks for all of the LGTMs
- A very special thanks to Andy Trick
- Further Questions: Dave Estes <cestes@codeaurora.org>