

Adding a Subtarget Support to LLVM in Five Minutes

Elvina Yakubova / elvinayakubova@gmail.com
Advanced Software Technology Lab, Huawei

HUAWEI TECHNOLOGIES CO., LTD.



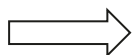
What to add to tablegen and how

- To solve the problem of describing the processor architecture, LLVM has a unified format for determining the processor properties required for the compiler which is called TableGen.

<https://llvm.org/docs/TableGen/>

llvm/lib/Target/AArch64/.td*

AArch64.td
AArch64Schedule.td
AArch64SchedA53.td
AArch64SchedA57.td
AArch64SchedA57WriteRes.td
AArch64SchedTsv110.td
AArch64SchedTsv110Details.td
AArch64CallingConvention.td
AArch64InstrAtomics.td
AArch64InstrFormats.td
AArch64InstrInfo.td
AArch64SchedPredicates.td
AArch64SystemOperands.td
AArch64RegisterBanks.td
AArch64RegisterInfo.td



```
def Tsv110Model : SchedMachineModel {  
  let IssueWidth = 4;           // Max micro-ops that may be scheduled per cycle.  
  let MicroOpBufferSize = 128; // Number of micro-ops that the processor may buffer for out-of-order execution.  
  let LoadLatency = 4;         // Optimistic load latency.  
  let MispredictPenalty = 14;  // Typical number of extra cycles the processor takes to recover from a branch misprediction.  
  let LoopMicroOpBufferSize = 16; // Number of micro-ops that the processor may buffer for optimized loop execution  
  let CompleteModel = 1;       // If you define a model for only a subset of instructions, you must clear this bit.  
  
  list<Predicate> UnsupportedFeatures = !listconcat(SVEUnsupported.F, PAUnsupported.F);  
}
```

What to add to tablegen and how

AArch64SchedTsv110.td

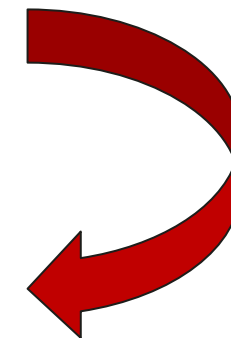
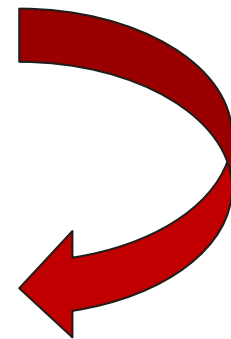
```
def Tsv110Model : SchedMachineModel {  
  let IssueWidth = 4;           // Max micro-ops that may be scheduled per cycle.  
  let MicroOpBufferSize = 128; // Number of micro-ops that the processor may buffer for out-of-order execution.  
  let LoadLatency = 4;         // Optimistic load latency.  
  let MispredictPenalty = 14;   // Typical number of extra cycles the processor takes to recover from a branch misprediction.  
  let LoopMicroOpBufferSize = 16; // Number of micro-ops that the processor may buffer for optimized loop execution  
  let CompleteModel = 1;       // If you define a model for only a subset of instructions, you must clear this bit.  
}
```

AArch64GenSubtargetInfo.inc

```
static const llvm::MCSchedModel Tsv110Model = {  
  4, // IssueWidth  
  128, // MicroOpBufferSize  
  4, // LoadLatency  
  14, // MispredictPenalty  
  ...  
  true, // CompleteModel  
  12, // Processor ID  
  Tsv110ModelProcResources,  
  Tsv110ModelSchedClasses,  
  ...  
};
```

AArch64MCTargetDesc.cpp

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



What to add to tablegen and how

- **Edit AArch64.td**
- **Create AArch64Sched<SubtargetName>.td**
- **Target description in AArch64Schedule.td**
- **Processor description in AArch64Sched<SubtargetName>.td**
 - Define SchedMachineModel
 - Define ProcResources
 - Map processor resources to default SchedWrites
 - Model Refining
- **Build**

What to add to tablegen and how

- **Edit AArch64.td**
- **Create AArch64Sched<SubtargetName>.td**
- **Target description in AArch64Schedule.td**
- **Processor description in AArch64Sched<SubtargetName>.td**
 - Define SchedMachineModel
 - Define ProcResources
 - Map processor resources to default SchedWrites
 - Model Refining
- **Build**

What to add to tablegen and how

AArch64.td

```
include "AArch64SchedTsv110.td" // File where you will describe your model
...
def ProcTSV110 : SubtargetFeature<"tsv110", "ARMProcFamily", "TSV110",
    "HiSilicon TS-V110 processors", [
    HasV8_2aOps,
    FeatureCrypto,
    FeatureCustomCheapAsMoveHandling,
    FeatureFPARMv8,
    FeatureFuseAES,
    FeatureNEON,
    FeaturePerfMon,
    FeaturePostRAScheduler,
    FeatureSPE,
    FeatureFullFP16,
    FeatureFP16FML,
    FeatureDotProd]>;
...
def : ProcessorModel<"tsv110", Tsv110Model, [ProcTSV110]>;
```

What to add to tablegen and how

- Edit AArch64.td
- Create AArch64Sched<SubtargetName>.td
- **Target description in AArch64Schedule.td**
- **Processor description in AArch64Sched<SubtargetName>.td**
 - Define SchedMachineModel
 - Define ProcResources
 - Map processor resources to default SchedWrites
 - Model Refining
- **Build**

What to add to tablegen and how

- Create scheduling categories for operands.

AArch64Schedule.td

```
| def WriteI      : SchedWrite; // ALU
| def WriteISReg  : SchedWrite; // ALU of Shifted-Reg
| def ReadI       : SchedRead; // ALU
| def ReadISReg   : SchedRead; // ALU of Shifted-Reg
```

- Associate scheduling categories to instructions.

AArch64InstrFormats.td

```
| class BaseCRC32<bit sf, bits<2> sz, bit C, RegisterClass StreamReg, SDPatternOperator OpNode, string asm>
| : I<(outs GPR32:$Rd), (ins GPR32:$Rn, StreamReg:$Rm),
|   asm, "\t$Rd, $Rn, $Rm", "",
|   [(set GPR32:$Rd, (OpNode GPR32:$Rn, StreamReg:$Rm))]>,
|   Sched<[WriteISReg, ReadI, ReadISReg]> {
| ...
| }
```

AArch64InstrInfo.td

```
| // CRC32
| def CRC32Brr : BaseCRC32<0, 0b00, 0, GPR32, int_aarch64_crc32b, "crc32b">;
| def CRC32Hrr : BaseCRC32<0, 0b01, 0, GPR32, int_aarch64_crc32h, "crc32h">;
| ...
```


What to add to tablegen and how

- Edit AArch64.td
- Create AArch64Sched<SubtargetName>.td
- Target description in AArch64Schedule.td
- **Processor description in AArch64Sched<SubtargetName>.td**
 - Define SchedMachineModel
 - Define ProcResources
 - Map processor resources to default SchedWrites
 - Model Refining
- Build

How to model pipeline and resources

- Define SchedMachineModel

AArch64SchedTsv110.td

```
def Tsv110Model : SchedMachineModel {  
  let IssueWidth = 4;           // Max micro-ops that may be scheduled per cycle.  
  let MicroOpBufferSize = 128; // Number of micro-ops that the processor may buffer for out-of-order execution.  
  let LoadLatency = 4;         // Optimistic load latency.  
  let MispredictPenalty = 14;  // Typical number of extra cycles the processor takes to recover from a branch misprediction.  
  let LoopMicroOpBufferSize = 16; // Number of micro-ops that the processor may buffer for optimized loop execution  
  let CompleteModel = 1;       // If you define a model for only a subset of instructions, you must clear this bit.  
                                // Otherwise, the scheduler considers an unmodelled opcode to be an error.  
  list<Predicate> UnsupportedFeatures = !listconcat(SVEUnsupported.F, PAUnsupported.F);  
}
```

- Define Processor Resources

AArch64SchedTsv110.td

```
let SchedModel = Tsv110Model in {  
  def TSV110UnitALU : ProcResource<1>; // ALU1  
  def TSV110UnitAB : ProcResource<2>; // ALU2/3/BRU1/2  
  def TSV110UnitMDU : ProcResource<1>; // Integer Multi-Cycle  
  def TSV110UnitFSU1 : ProcResource<1>; // FP/ASIMD1  
  def TSV110UnitFSU2 : ProcResource<1>; // FP/ASIMD2  
  def TSV110UnitLdSt : ProcResource<2>; // Load/Store0/1  
}
```

- Map processor resources to default SchedWrites

AArch64SchedTsv110.td

```
def : WriteRes<WriteISReg, [TSV110UnitMDU]> { let Latency = 2; }  
def : WriteRes<WriteIReg, [TSV110UnitMDU]> { let Latency = 2; }  
def : WriteRes<WriteST, [TSV110UnitLdSt]> { let Latency = 1; }  
  
// Forwarding information  
def : ReadAdvance<ReadISReg, 0>;  
def : ReadAdvance<ReadIM, 0>;  
def : ReadAdvance<ReadIMA, 2, [WriteIM32, WriteIM64]>;
```

How to model pipeline and resources

- Model refining

Override target default operations with specific processor behavior

AArch64SchedTsv110Details.td

```
// Cryptography Extensions
// -----

def TSV110Wr_2cyc_1F      : SchedWriteRes<[TSV110UnitF]>    { let Latency = 2; }
def TSV110Wr_2cyc_1FSU1 : SchedWriteRes<[TSV110UnitFSU1]> { let Latency = 2; }
def TSV110Wr_3cyc_1FSU1 : SchedWriteRes<[TSV110UnitFSU1]> { let Latency = 3; }
def TSV110Wr_5cyc_1FSU1 : SchedWriteRes<[TSV110UnitFSU1]> { let Latency = 5; }

def : InstRW<[TSV110Wr_3cyc_1FSU1], (instregex "^AES[DE]")>;
def : InstRW<[TSV110Wr_2cyc_1FSU1], (instregex "^SHA1SU1")>;
def : InstRW<[TSV110Wr_2cyc_2F],    (instregex "^SHA1(H|SU0)")>;
def : InstRW<[TSV110Wr_5cyc_1FSU1], (instregex "^SHA1[COMP]")>;
def : InstRW<[TSV110Wr_2cyc_1FSU1], (instregex "^SHA256SU0")>;
def : InstRW<[TSV110Wr_3cyc_1FSU1], (instregex "^SHA256SU1")>;
def : InstRW<[TSV110Wr_5cyc_1FSU1], (instregex "^SHA256(H|H2)")>;
```

How to measure performance

- **llvm test-suite**

<https://llvm.org/docs/TestSuiteGuide.html>

test-suite structure:

- ❑ *SingleSource* – contains programs that consists of a single source file (small benchmarks).
- ❑ *MultiSource* – contains entire programs with multiple source files (large benchmarks and whole applications).
- ❑ *MicroBenchmarks* – programs that use google-benchmark library. They define functions that are executed several times until the measurement results are statistically significant.
- ❑ *External Suites* – contains support for running tests which cannot be directly distributed with the test-suite (ex. SPEC)
- ❑ *Bitcode* – tests that are written in LLVM bitcode.
- ❑ *CTMark* – set of compile time benchmarks to measure compile time.

How to measure performance

- **llvm test-suite**

<https://llvm.org/docs/TestSuiteGuide.html>

test-suite structure:

- ❑ *SingleSource* – contains programs that consists of a single source file (small benchmarks).
- ❑ *MultiSource* – contains entire programs with multiple source files (large benchmarks and whole applications).
- ❑ *MicroBenchmarks* – programs that use google-benchmark library. They define functions that are executed several times until the measurement results are statistically significant.
- ❑ *External Suites* – contains support for running tests which cannot be directly distributed with the test-suite (ex. SPEC)
- ❑ *Bitcode* – tests that are written in LLVM bitcode.
- ❑ *CTMark* – set of compile time benchmarks to measure compile time.

From test-suite-build directory:

```
# configuration
cmake \
  -DCMAKE_C_COMPILER:FILEPATH=clang \
  -DTEST_SUITE_BENCHMARKING_ONLY=ON \
  -DTEST_SUITE_RUN_BENCHMARKS=ON \
  -DCMAKE_C_FLAGS="-mcpu=tsv110"
  ../test-suite

# build the benchmarks
make

# run the tests with lit
llvm-lit -v -j 1 -o res.json

# Show and compare result files
test-suite/utils/compare.py res.json
```

How to measure performance

- *llvm test-suite*

<https://llvm.org/docs/TestSuiteGuide.html>

The result for each test is displayed as:

```
PASS: test-suite :: MultiSource/Benchmarks/Rodinia/pathfinder/pathfinder.test (123 of 311)
***** TEST 'test-suite ::
MultiSource/Benchmarks/Rodinia/pathfinder/pathfinder.test' RESULTS *****
compile_time: 1.5648
exec_time: 0.3917
hash: "23c06845d751c8861195dd7de31687cf"
link_time: 0.0247
size: 71808
size..bss: 1392
size..comment: 122
size..data: 16
...
size..hash: 56
size..init: 20
size..init_array: 8
size..interp: 27
size..note.ABI-tag: 32
size..plt: 128
size..rodata: 8
size..text: 3084
*****
```

How to measure performance

- *llvm test-suite*

<https://llvm.org/docs/TestSuiteGuide.html>

The result for each test is displayed as:

```
PASS: test-suite :: MultiSource/Benchmarks/Rodinia/pathfinder/pathfinder.test (123 of 311)
***** TEST 'test-suite ::
MultiSource/Benchmarks/Rodinia/pathfinder/pathfinder.test' RESULTS *****
compile_time: 1.5648
exec_time: 0.3917
hash: "23c06845d751c8861195dd7de31687cf"
link_time: 0.0247
size: 71808
size..bss: 1392
size..comment: 122
size..data: 16
...
size..hash: 56
size..init: 20
size..init_array: 8
size..interp: 27
size..note.ABI-tag: 32
size..plt: 128
size..rodata: 8
size..text: 3084
*****
```

Compare results from different launches:

Metric: exec_time
Geomean difference

	Res1.json	Res2.json	Res3.json
count	703	703	703
mean	2207.896468	2207.308501	2206.653598
std	20729.320121	20732.956600	20652.590397
0%	0.000000	0.000000	0.000000
25%	1.263045	1.263617	1.263912
50%	9.353500	9.368500	9.405600
75%	548.393830	547.112793	547.850836
max	364026.84000	363886.43500	362920.14500

How to measure performance

- llvm-mca

<https://llvm.org/docs/CommandGuide/llvm-mca.html>

llvm-mca is a performance analysis tool that uses information available in LLVM (e.g. scheduling models) to statically measure the performance of machine code on a specific CPU.

Can be used to:

- Predict performance of code.
- Diagnose potential performance issues.
- Test machine scheduling models.

```
Iterations: 100
Instructions: 100
Total Cycles: 203
Total uOps: 200

Dispatch Width: 4
uOps Per Cycle: 0.99
IPC: 0.49
Block RThroughput: 1.0
```

```
Timeline view:
Index      0123456789      012
[0,0]      DeeER. . . sha1su0 v0.4s, v1.4s, v2.4s
[1,0]      D==eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[2,0]      .D===eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[3,0]      .D====eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[4,0]      . D=====eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[5,0]      . D=====eeER. . sha1su0 v0.4s, v1.4s, v2.4s
[6,0]      . D=====eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[7,0]      . D=====eeER . . sha1su0 v0.4s, v1.4s, v2.4s
[8,0]      . D=====eeER . sha1su0 v0.4s, v1.4s, v2.4s
[9,0]      . D=====eeER sha1su0 v0.4s, v1.4s, v2.4s

Average wait times (based on the timeline view):
[0]: Executions
[1]: Average time spent waiting in a scheduler's queue
[2]: Average time spent waiting in a scheduler's queue while ready
[3]: Average time elapsed from WB until retire stage

0.      [0]  [1]  [2]  [3]
         10  8.0  0.1  0.0      sha1su0 v0.4s, v1.4s, v2.4s
```


Thank you

Elvina Yakubova
elvinayakubova@gmail.com