Machine Scheduler
Fine grain resource allocation using ResourceSegments

Adam Nemet, Francesco Petrogalli (speaker), Francis Visoiu-Mistrih - Apple
What is this all about

- MachineScheduler and SchedMachineModel
- No InstrItineraries
- Representation of hardware resources in the SchedMachineModel
- Improved estimates of execution traces
  - Better scheduling
- Ongoing effort
Background information
Definitions
Definitions

Instruction is

*Ready @<cycle>*

All input data needed by an instruction is ready.
Definitions

Instruction is **Ready @<cycle>**

All input data needed by an instruction is ready

ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2
Definitions

Instruction is **Ready @<cycle>**

All input data needed by an instruction is ready

ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2

Instruction is **Available @<cycle>**

All hardware resources that execute the instruction are available
Definitions

Instruction is

*Ready* @<cycle>

All input data needed by an instruction is ready

Instruction is

*Available* @<cycle>

All hardware resources that execute the instruction are available

ADD r2, r1, r0
ADD r5, r4, r3
MUL r6, r5, r2
All instructions used in this presentation are READY
ADD r2, r0, r1
ADD r4, r3, r2
Focus on structural hazards

ADD r2, r1, r0
SUB r5, r4, r3
Instructions breakdown
Instructions breakdown

Sequence of stages
Instructions breakdown

Sequence of stages

Fetch
Instructions breakdown

Sequence of stages

Fetch  Decode
Instructions breakdown

Sequence of stages

Fetch → Decode → Execute
Instructions breakdown

Sequence of stages

Fetch → Decode → Execute → ...

17
Instructions used in this talk

**ADD**

- Fetch
- Decode
- Execute ADD
- ...
- Retire

**MADD**

- Fetch
- Decode
- Execute MADD
- ...
- Retire
Let’s schedule some code!
The stage “Execute MADD” can process one instruction at a time
The stage “Execute MADD” can process one instruction at a time

MADD r3, r2, r1, r0
MADD r7, r6, r5, r4
MADD r11, r10, r9, r8
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time
The stage “Execute MADD” can process one instruction at a time
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
The stage “Execute MADD” can process one instruction at a time.
Focus on the instruction-specific resources
Focus on the instruction-specific resources

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r3, r2, r1, r0</td>
<td>Execute MADD</td>
<td>...</td>
<td>...</td>
<td>Execute MADD</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>Execute MADD</td>
</tr>
<tr>
<td>MADD r7, r6, r5, r4</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>MADD r11, r10, r9, r8</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Pipelined execution
Break up the execution in separate stages

Hardware feature
Pipelined resources: faster execution!

MADD r3, r2, r1, r0
MADD r7, r6, r5, r4
MADD r11, r10, r9, r8

0 1 2 3 4 5 6 7 8

Execute MADD
Multiplier Adder
Multiplier Adder
Multiplier Adder
What happens when pipeline execution shares functional units?
Reminder
Focus on the execution units

ADD

Adder (1 cycle)

MADD

Multiplier (2 cycles) Adder (1 cycle)
Execution with a shared Adder

MADD r2, r1, r0
ADD r5, r4, r3
ADD r8, r7, r6
Execution with a shared Adder
Execution with a shared Adder

MADD r2, r1, r0
Multiplier
Adder
ADD r5, r4, r3
Adder
ADD r8, r7, r6
Execution with a shared Adder
Execution with a shared Adder
Postpone execution (stall)
LLVM’s representation of resources

ADD

MADD

Multiplier (2 cycles) ➔ Adder (1 cycle)
LLVM’s representation of resources

**ADD**
- Adder (1 cycle)

**MADD**
- Multiplier (2 cycles) → Adder (1 cycle)

**TableGen description**
```python
def : WriteRes<WriteADD, [Adder]> {
    let ResourceCycles = [ 1];
}
```
**LLVM’s representation of resources**

**TableGen description**

```ruby
def : WriteRes<WriteADD, [Adder]> {
    let ResourceCycles = [ 1 ];
}
```

### ADD
- **Adder** (1 cycle)

### MADD
- **Multiplier** (2 cycles)
- **Adder** (1 cycle)
LLVM’s representation of resources

**ADD**
- Adder (1 cycle)

**MADD**
- Multiplier (2 cycles)
  - Adder (1 cycle)

**TableGen description**

```cpp
def : WriteRes<WriteADD, [Adder]> {
    let ResourceCycles = [ 1 ];
} def : WriteRes<WriteMADD, [Multiplier, Adder]> {
    let ResourceCycles = [ 2, 3 ];
}
```
LLVM’s representation of resources

**TableGen description**

```python
def WriteRes<WriteADD, [Adder]>(
    ResourceCycles = [ 1];
}
def WriteRes<WriteMADD, [Multiplier, Adder]>(
    ResourceCycles = [ 2, 3];
```
LLVM's representation of resources

The Adder resource is overbooked for 2 extra cycles in the MADD instruction

TableGen description

```cpp
def : WriteRes<WriteADD, [Adder]> {  
  let ResourceCycles = [ 1];  
}
def : WriteRes<WriteMADD, [Multiplier, Adder]> {  
  let ResourceCycles = [ 2, 3];  
}
```
LLVM estimation of execution with shared resources.
What LLVM estimates

MADD r2, r1, r0
ADD r5, r4, r3
ADD r8, r7, r6
What LLVM estimates

- MADD r2, r1, r0
- ADD r5, r4, r3
- ADD r8, r7, r6
What LLVM estimates

MADD r2, r1, r0
Multiplier
Adder
ADD r5, r4, r3
Adder
ADD r8, r7, r6
What LLVM estimates

MADD r2, r1, r0
Multiplier
Adder

ADD r5, r4, r3
Adder

ADD r8, r7, r6
What LLVM estimates

MADD r2, r1, r0
Multiplier
Adder

ADD r5, r4, r3
Adder

ADD r8, r7, r6
Adder
Overbooking of resources leads to longer traces

What LLVM estimates

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r2, r1, r0</td>
<td>Multiplier</td>
<td>Adder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r5, r4, r3</td>
<td>Adder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r8, r7, r6</td>
<td>Adder</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What hardware does

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r2, r1, r0</td>
<td>Multiplier</td>
<td>Adder</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r5, r4, r3</td>
<td>Adder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r8, r7, r6</td>
<td>Adder</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Overbooking of resources leads to longer traces

**What LLVM estimates**

- MADD r2, r1, r0
- ADD r5, r4, r3
- ADD r8, r7, r6

**What hardware does**

- MADD r2, r1, r0
- ADD r5, r4, r3
- ADD r8, r7, r6
Resource Segments

Fine grain resolution of resource usage
def : WriteRes<WriteADD, [Adder]> { let ResourceCycles = [ 1]; }
def : WriteRes<WriteMADD, [Multiplier, Adder]> { let ResourceCycles = [ 2, 3]; }
ResourceSegments

Thinking in terms of intervals, open on the right: \([A, B)\)

def WriteRes<WriteADD, [Adder]> { 
    let ResourceCycles = [1];
}
def WriteRes<WriteMADD, [Multiplier, Adder]> { 
    let ResourceCycles = [2, 3];
}
def WriteRes<WriteADD, [Adder]> {  
let ResourceCycles = [ 1];  
}
def WriteRes<WriteMADD, [Multiplier, Adder]> {  
let ResourceCycles = [ 2, 3];  
}
def : WriteRes<WriteADD, [Adder]> { 
    let ResourceCycles = [ 1]; 
}
def : WriteRes<WriteMADD, [Multiplier, Adder]> { 
    let ResourceCycles = [ 2, 3]; 
}
Intermission

Advertise a new feature
From fancy tables...

MADD r2, r1, r0
Multiplier
Adder
ADD r5, r4, r3
Adder
ADD r8, r7, r6
Adder
...to text tables!

*** Final schedule for %bb.0 ***

* Schedule table (TopDown):
  
  i: issue
  
  x: resource booked

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r2, r1, r0</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplier</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ADD r5, r4, r3</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r8, r7, r6</td>
<td></td>
<td></td>
<td>i</td>
<td>x</td>
</tr>
</tbody>
</table>
Debug messages generated by the compiler!

```plaintext
*** Final schedule for %bb.0 ***
* Schedule table (TopDown):
  i: issue
  x: resource booked

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r2, r1, r0</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplier</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ADD r5, r4, r3</td>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r8, r7, r6</td>
<td></td>
<td></td>
<td>i</td>
<td>x</td>
</tr>
</tbody>
</table>
```

LIT unit tests for resource usage in scheduling models

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>MADD r2, r1, r0</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiplier</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r5, r4, r3</td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD r8, r7, r6</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>Adder</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
Implementation
What changes in the code
TableGen representation and MachineScheduler

• TableGen:
  • `list<int> StartAtCycle = []`; added to the WriteRes class;
  • Backend changes in `llvm/utils/TableGen/SubtargetEmitter.cpp`
What changes in the code
TableGen representation and MachineScheduler

• TableGen:
  • `list<int> StartAtCycle = []`; added to the WriteRes class;
  • Backend changes in `llvm/utils/TableGen/SubtargetEmitter.cpp`

• MachineScheduler:
  • Data structure to handle intervals
  • New fine grain bookkeeping algorithm
Fine grain bookkeeping

Keeping track of resource intervals across the schedule
Current algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10

New algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10
Current algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10

New algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10

Legend

Multiplier
Adder

Assumed
Seen

Last Seen

All Seen
Current algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10

Last Seen

Legend
Assumed
Seen
Multiplier
Adder

New algorithm

ADD r2, r1, r0
ADD r5, r4, r3
MADD r9, r8, r7, r6
ADD r12, r11, r10

All Seen
Current algorithm

ADD r2, r1, r0
Adder

ADD r5, r4, r3
Adder

MADD r9, r8, r7, r6
Multiplier

ADD r12, r11, r10
Adder

New algorithm

ADD r2, r1, r0
Adder

ADD r5, r4, r3
Adder

MADD r9, r8, r7, r6
Multiplier

ADD r12, r11, r10
Adder

Legend
Assumed
Seen

Multiplier
Adder
Finds the gap in the disjoint interval!

**Current algorithm**

0 1 2 3

ADD r2, r1, r0
Adder

ADD r5, r4, r3
Adder

MADD r9, r8, r7, r6
Multiplier
Adder

ADD r12, r11, r10
Adder

...  

**Last Seen**

Multiplier
Adder

**New algorithm**

0 1 2 3

ADD r2, r1, r0
Adder

ADD r5, r4, r3
Adder

MADD r9, r8, r7, r6
Multiplier
Adder

ADD r12, r11, r10
Adder

...  

**All Seen**

Multiplier
Adder

Legend
- Assumed
- Seen
Current algorithm

ADD r2, r1, r0
Adder
ADD r5, r4, r3
Adder
MADD r9, r8, r7, r6
Multiplier
ADD r12, r11, r10
Adder

Last Seen
Multiplier
Adder

New algorithm

ADD r2, r1, r0
Adder
ADD r5, r4, r3
Adder
MADD r9, r8, r7, r6
Multiplier
ADD r12, r11, r10
Adder

All Seen
Multiplier
Adder
Better estimate of execution.

Current algorithm

New algorithm
Performance improvements
Example 1: from 25 cycles to 12 cycles
Top-down scheduling

<table>
<thead>
<tr>
<th>Cycle</th>
<th>ADD</th>
<th>SUB</th>
<th>SLL</th>
<th>MUL</th>
<th>SRL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

x: resource booked
i: issue

$\text{bb.0:} \quad \begin{align*}
\text{liveins: } & x9, x11, x13, x15, x17, x19 \\
& x10 = \text{ADD} \ x9, x9 \\
& x12 = \text{SUB} \ x11, x11 \\
& x16 = \text{SLL} \ x15, x15 \\
& x14 = \text{MUL} \ x13, x13 \\
& x18 = \text{SRL} \ x17, x17 
\end{align*}$
### Example 2: from 17 cycles to 7 cycles

**Bottom-up scheduling**

<table>
<thead>
<tr>
<th>Cycle</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX3</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>ResX3</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLL</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX3</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>SLL</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX0</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRL</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX3</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX2</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV.</td>
<td></td>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResX1</td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### bb.0:

- **liveins**: $x9, $x11, $x13, $x15, $x17, $x19
- $x10 = ADD $x9, $x9
- $x12 = SUB $x11, $x11
- $x14 = MUL $x13, $x13
- $x16 = SLL $x15, $x15
- $x18 = SRL $x17, $x17
- $x20 = DIV $x19, $x19
## Average improvements

### Artificial test cases (LIT)

<table>
<thead>
<tr>
<th>TEST</th>
<th>TOP-DOWN (cycles)</th>
<th>BOTTOM-UP (cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Current</td>
<td>New</td>
</tr>
<tr>
<td>test-001</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>test-002</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>test-003</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>test-004</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>test-005</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>test-006</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>test-007.A</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>test-007.B</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>test-008</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>test-009</td>
<td>9</td>
<td>5</td>
</tr>
<tr>
<td>test-010</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>test-012</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td>134</td>
<td>72</td>
</tr>
</tbody>
</table>

New/Current = **0.53731**  
New/Current = **0.54362**
Recap
Better scheduling

...and testing!

ResourceSegments + Fine Grain Bookkeeping → Better scheduling!
Better scheduling

...and testing!

ResourceSegments + Fine Grain Bookkeeping → Better scheduling!

llc -misched-dump-schedule-trace
What’s next
Adoption steps

• All current models are defaulted with StartAtCycle = [0, ..., 0];

• Aim at replacing the current bookkeeping in the machine scheduler with the new one.

• Bit switch in the schedule model class to enable the new codepath.

• Further investigations:
  • Few CodeGen issues (it seems to find gaps that couldn’t be found before)
  • Compile time (threshold of 10 intervals per resource).

• Work is ongoing, but WIP patches are up for review / feedback / try out
Reviews on Phabricator
Feedback is welcome!

• D150310: Adding StartAtCycle to WriteRes (NFC)
• D150311: Schedule traces in debug
• D150312: Modify MachineScheduler to use StartAtCycle
Thank you!

Questions?