



Using LLVM in the presence of timing constraints

Dave Lacey

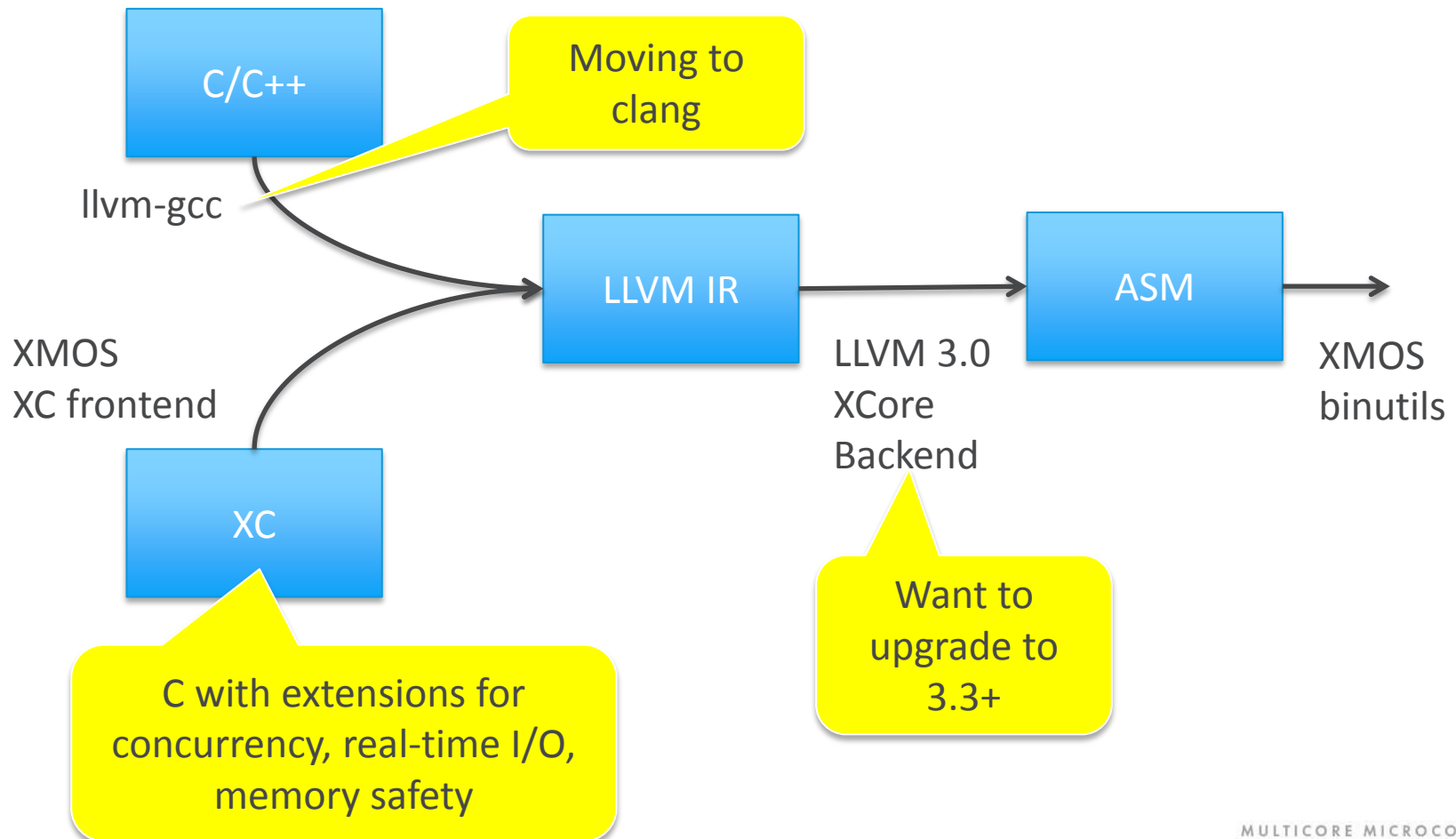
LLVM European Developers Meeting

30th April 2013



XMOS and LLVM

- XMOS uses LLVM to implement C + XC compilers:





Real-time WCET constraints

- Code for our devices is hard real-time

```
port p, q;
time_t t1, t2;
...
wait_for_edge_and_timestamp(p, &t1);
...
while (!cond) {
    ...
    output_signal_and_timestamp(q, &t2);
    timing_assert((t2 - t1) < 270);
    ...
    wait_for_edge_and_timestamp(p, &t1);
}
```

If the time between these points is less than 270ns then the program does not work

- We have an analysis tool to check these constraints (XTA)



What's the problem?

- LLVM has many optimizations
- In general, these optimization aim to improve average execution time not worst case execution time
- In general, these optimization aim to make the whole function (or perhaps loop within the function) faster - no prioritization between execution paths
- Optimizations can make things much worse (from a WCET perspective)



Example: Scheduling

- Scheduling can mess things up:

```
port p, q;
time_t t1, t2;
...
wait_for_edge_and_timestamp(p, &t1);
...
while (!cond) {
    [code sequence 1]

    output_signal_and_timestamp(q, &t2);
    timing_assert((t2 - t1) < 270);
    [code sequence 2]
    [code sequence 3]
    wait_for_edge_and_timestamp(p, &t1);
}
```



**Constraints are met.
The program works!**



Example: Scheduling

- Scheduling can mess things up:

```
port p, q;
time_t t1, t2;
...
wait_for_edge_and_timestamp(p, &t1);
...
while (!cond) {
    [code sequence 1]
    [code sequence 2]
    output_signal_and_timestamp(q, &t2);
    timing_assert((t2 - t1) < 270);

    [code sequence 3]
    wait_for_edge_and_timestamp(p, &t1);
}
```



**Takes too long.
Broken!**

Example: Invariant hoisting

- Loop invariant hoisting can mess things up:

```
port p, q;
time_t t1, t2;
...
wait_for_edge_and_timestamp(p, &t1);
...

while (!cond) {
    [code sequence 1]
    output_signal_and_timestamp(q, &t2);
    timing_assert((t2 - t1) < 270);
    [code sequence 2]
    [code sequence 3]
    wait_for_edge_and_timestamp(p, &t1);
}
```



**Constraints are met.
The program works!**

Example: Invariant hoisting

- Loop invariant hoist can mess things up:

```
port p, q;
time_t t1, t2;
...
wait_for_edge_and_timestamp(p, &t1);
...
[code sequence 2]
while (!cond) {
    [code sequence 1]
    output_signal_and_timestamp(q, &t2);
    timing_assert((t2 - t1) < 270);

    [code sequence 3]
    wait_for_edge_and_timestamp(p, &t1);
}
```



**Takes too long.
Broken!**



What are we going to do?

- What are we going to do....
- Um...



Some hopes

- Most optimizations are OK
- We can sort most of this out in the scheduler
- ... but that requires a scheduler that isn't just a basic block scheduler
- Need to avoid a big fork. Most platforms/code do not care about this as much so cannot rewrite LLVM to be “worst case constraint aware” everywhere.
- Limiting optimizations that cause problems is hopefully a matter of tuning rather than rewriting