Embedded Operating System Optimization through Floating to Fixed Point Compiler Transformation

D. Cattaneo, A. Di Bello, S. Cherubin, F. Terraneo, G. Agosta

POLITECNICO MILANO 1863

HEAP LAB

{daniele3.cattaneo, antonio.dibello}@mail.polimi.it
{stefano.cherubin, federico.terraneo, giovanni.agosta}@polimi.it

We propose a self-contained compiler transformation pass implemented within LLVM. We demonstrate the effectiveness of our solution on a set of ARM microcontrollers, both with and without hardware support for floating point arithmetic.

MIOSIX OPERATING SYSTEM

MIOSIX is written almost entirely in C++, its uses include:

- wireless sensor network nodes
- development and evaluation boards
- wearable devices such as smartwatches
- control boards for experimental sounding rockets
- platform for academic research to design innovative task scheduling algorithms [1]
- clock synchronization solutions [2]

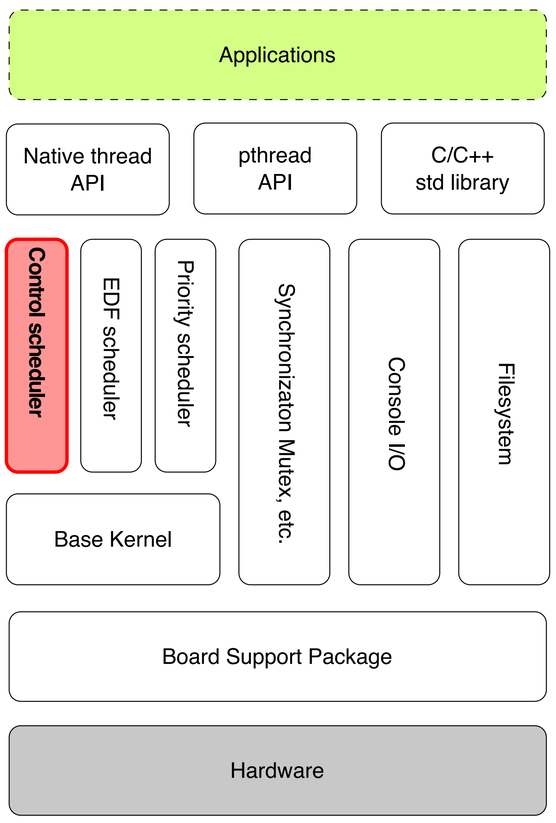
CONTROL-THEORETICAL SCHEDULER

MIOSIX features a task scheduler based on control theory.

At every scheduling round, a control-theoretical regulator is executed to compute the actual burst time for each thread in the next round. It exploits the time actually used by the task in the previous round as feedback.

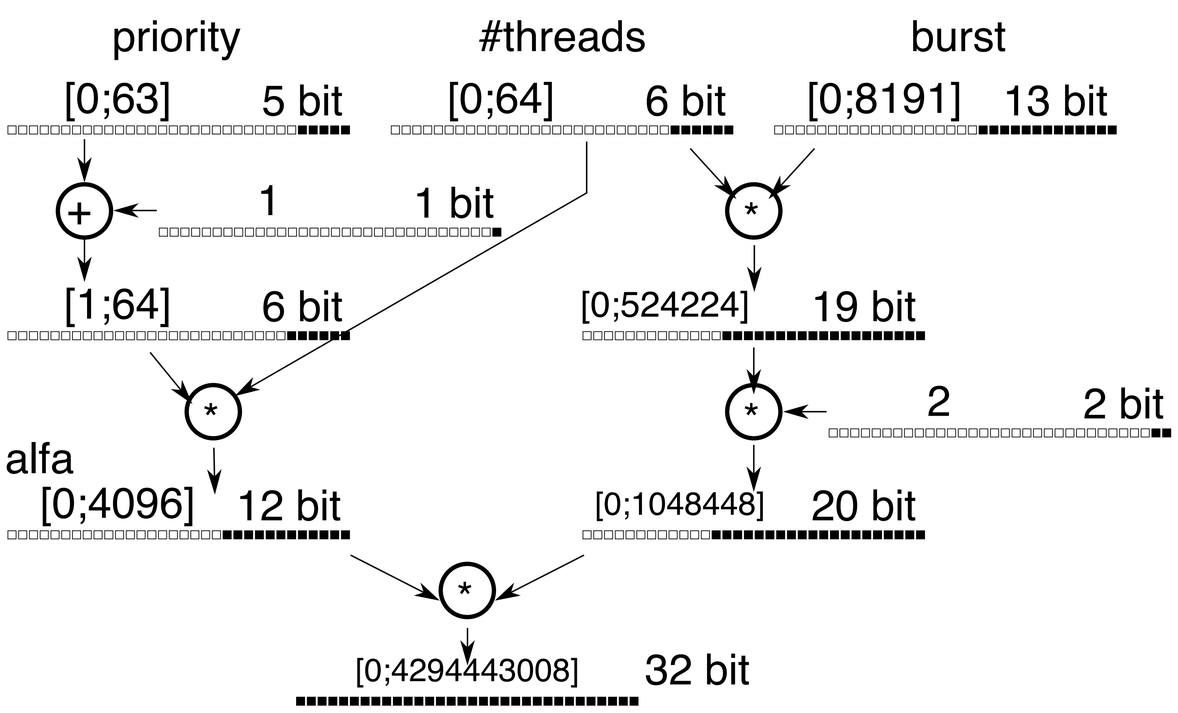
The implementation of the regulator uses floating point variables.

Given the default OS configuration it is possible to compute the initial range of values for each of the floating point variables.

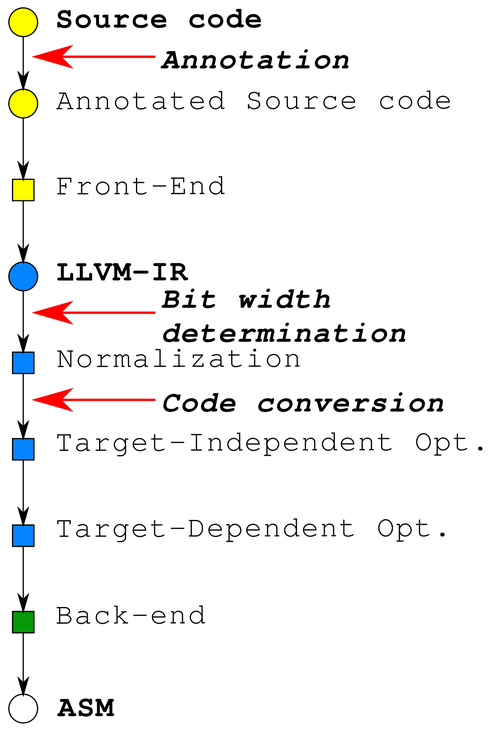


DATA FLOW ANALYSIS (EXAMPLE)

Data flow analysis of the floating point values used in a fragment of the MIOSIX task scheduler. For each node we report the range and the minimum data width required.



LLVM CONVERSION PASS



Our proposed transformation exploits only on the compiler middle-end APIs which can be invoked through pluggable external modules called passes. The programmer **annotates the source** code via custom *annotate* attributes on floating point variables whose type they wish to convert to fixed point.

Attributes are parsed by the vanilla compiler front-end, which transfers them to the middle-end.

Our compiler pass **collects those annotations** and properly propagates them to intermediate values.

It **creates instructions** based on fixed point arithmetic equivalent to the original code.

We **compile** the converted code for the target architecture and we integrate the object code into the MIOSIX build system.

ANNOTATION EXAMPLE

Variables and computations which are converted to fixed point.
Connections between them highlight the operations affected by the conversion.

It is possible to specify the size of the integer and fractional part of the representation.

```
float a __attribute((annotate("force_no_float")));
int b = 98;
a = b * 2.0;
a += 10.0;
float c __attribute((annotate("no_float 12 20")));
c = function1(b) + 3.5;
a += c * 2.0;
```

The `no_float` and `force_no_float` annotations indicate which variables have to be converted to fixed point. The conversion propagates to related computations in different ways: `force_no_float` entails the conversion of the dependencies, while `no_float` propagates only to intermediate values.

RESULTS

We compare our solution (*pass* version) against these alternatives:

- The original floating point implementation (*reference*)
- Fixed point C++ header library [3] (*C++lib*)
- Manual porting from floating point to integer arithmetic (*manual*)

We compile the aforementioned versions both with the default Miosix toolchain (customized version of GCC) and clang 4.0.0

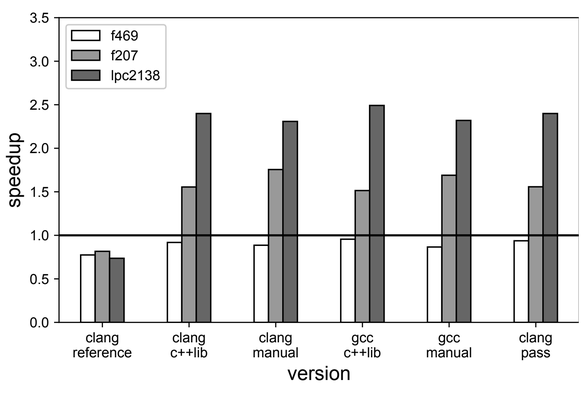
f207: STM3220G-EVAL board
120MHz ARM Cortex M3 **without** hw floating point support

f469: STM32F469I-DISCO board
168MHz ARM Cortex M4 **with** hw floating point support

lpc2138: development board
59MHz ARM7TDMI **without** hw floating point support

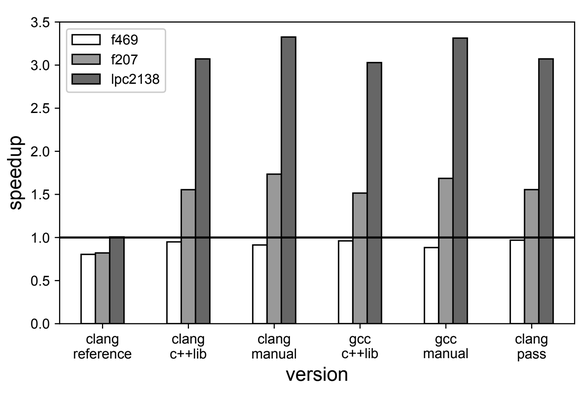
SPEEDUP Hartstone

Speedup w.r.t. GCC reference version of IRQrunRegulator function measured on the Hartstone uniprocessor benchmark suite [4]



SPEEDUP MiBench

Speedup w.r.t. GCC reference version of IRQrunRegulator function measured on a subset of benchmarks from the MiBench suite [5]



REFERENCES

[1] M. Maggio, F. Terraneo, and A. Leva
Task scheduling: a control-theoretical viewpoint for a general and flexible solution
Trans. on Embedded Computing Systems, vol. 13, no. 4, pp. 1–22, 2014.

[2] F. Terraneo, A. Leva, S. Seva, M. Maggio, and A. V. Papadopoulos
Reverse flooding: Exploiting radio interference for efficient propagation delay compensation in WSN clock synchronization
2015 IEEE Real-Time Systems Symposium

[3] S. Cherubin. G. Agosta, I. Lasri. E. Rohou, O. Sentieys
Implications of Reduced-Precision Computations in HPC: Performance, Energy and Error
International Conference on Parallel Computing, Sep 2017

[4] N. H. Weiderman and N. I. Kamenoff
Hartstone uniprocessor benchmark: Definitions and experiments for real-time systems
Real-Time Systems, vol. 4, no. 4, pp. 353–382, Dec 1992

[5] M. R. Guthaus et al.
Mibench: A free, commercially representative embedded benchmark suite
Fourth Annual IEEE International Workshop on Workload Characterization, Dec 2001, pp. 3–14.

ACKNOWLEDGEMENTS

This work is supported by the European Union's Horizon 2020 research and innovation programme, under grant agreement No 671623, FET-HPC ANTAREX.

ANTAREX 10^18