# Code Transformation and Optimization
# 2018 Course Projects

G. Agosta, S. Cherubin, A. Di Federico, F. Terraneo

http://www.heaplab.deib.polimi.it

## OpenCL C

OpenCL C is (mostly) a restriction of C, with some extensions to support the memory model of OpenCL, as well as specialized libraries.
In MANGO, we aim at supporting OpenCL on NU+ and PEAK, two custom parallel architectures.

## Project goals (one per project assigned)

- Analyse and implement support for OpenCL C for NU+
- Analyse and implement support for OpenCL C for PEAK

# GEOPM Extension

Automated generation of instrumentation ◇ Giovanni Agosta

## Global Extensible Open Power Manager

GEOPM is a monitoring tool for HPC computing systems, that aims at providing monitoring suitable for deployment on real HPC infrastructures.

## Project goals

- Develop a way to automatically generate the necessary instrumentation at application, based on static analysis.
- Determine the points in the code where notifications must be inserted.
- Perform the insertion of calls to `libgeopm`

## References

- `https://geopm.github.io/`
- `https://link.springer.com/content/pdf/10.1007/978-3-319-58667-0_21.pdf`

# LLVM support for Miosix

Federico Terraneo

Miosix

## Miosix is an OS kernel for microcontrollers

- Currently targets ARM processors
- Currently only works with GCC
- To support thread-safety, requires patches to the compiler built-ins library

## Project goals

- Compile the kernel using LLVM
    - requires building LLVM for ARM with the correct C/C++ standard library
- Patch llvm to support thread-safety in Mioisx

# Efficient Non-Standard Numeric System C++ Representations

Continued Fractions ⋄ Stefano Cherubin

A continued fraction $[a_1; a_2; a_3; \ldots]$ is an expression of the form:

$$a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \ldots}}$$

## Project goals

- Implement a C++ library with operators to make its use convenient.
- Use finite continued fractions to represent numbers.
- Evaluate their efficiency on a set of benchmarks (to be defined).

## References

- `https://github.com/skeru/fixedpoint`
- `https://crypto.stanford.edu/pbc/notes/contfrac/`

The Logarithmic Number System (LNS) is a representation of real numbers such that a number $x$ is represented as $\{s(x), log_b(|x|)\}$, where $s(x) = 0$ if $x > 0$, $s(x) = 1$ otherwise.

## Project goals

- Implement a C++ library with operators to make its use convenient.
- Use floating point LNS to represent numbers.
- Evaluate their efficiency on a set of benchmarks (to be defined).

## References


- `https://github.com/skeru/fixedpoint`
- `http://coep.vlab.co.in/?sub=29&brch=88&sim=1353&cnt=1`

`llvm-mca` and `llvm_sim` are novel static performance analysis tools implemented in LLVM.

## Project goals

- Apply static analysis to different versions of machine code to understand which version can provide better performance.
- Exploit `llvm-mca` and `llvm_sim` on top of a compiler optimization pass to measure possible benefit of the optimizations on different architectures.

## References

- `https://llvm.org/docs/CommandGuide/llvm-mca.html`
- `https://github.com/google/EXEgesis/tree/master/llvm_sim`

# Static Analysis

Value Range analysis ◇ Stefano Cherubin

Value Range Analysis aims at understanding through a data-flow analysis approach the evolution of the computed values given the possible initial ranges of the input data.

## Project goals

- Implement a value range analysis in LLVM to estimate the range of actual values each variable can assume.
- Apply this analysis to the problem of bit partitioning in fixed point arithmetic.
- Evaluation should be performed on approximate computing benchmarks

## References

- `http://axbench.org`

Darulova & Kuncak propose a method for estimating the error imposed by the use of a reduced precision data type for storage and computation.

## Project goals

- Implement the error estimation algorithm proposed in Section 5 of their paper.
- Evaluation should be performed on approximate computing benchmarks.

## References


- E.Darulova,V.Kuncak,"TowardsacompilerforReals",doi: 10.1145/3014426
- http://axbench.org

Performing a diff between binaries is useful in general, and in particular for plagiarism detection purposes.

## Project goals

- Build a robust algorithm working on LLVM IR recovered by rev.ng to detect similar functions and assign a similarity score.
- The starting point would be looking at how BinDiff works

## References

- `https://rev.ng`
- `https://www.zynamics.com/bindiff.html`

Currently rev.ng takes a full binary, lifts it to LLVM IR and then recompiles it fully.

## Project goals

- Let the user choose only a subset of functions that they are interested in modifying, let them change them (e.g., flip a condition, inject tracing or whatnot) and then recompile exclusively that function and patch it back in the original binary.
- The project involves in particular LLVM's JIT engine.

## References

- `https://rev.ng`
- `https://llvm.org/docs/tutorial/BuildingAJIT1.html`

A binary lifter is a piece of software that, given a buffer of bytes, it interprets it as executable code and returns an IR. `rev.ng` provides such a library based on QEMU.

## Project goals

- Cleaning up the library, porting it to the most recent version of QEMU and push to upstream our changes to the official QEMU project.

## References

- `https://rev.ng`
- `https://llvm.org/docs/tutorial/BuildingAJIT1.html`

When reverse engineering binary code, a key issue is the reconstruction of automatic variables.

## Project goals

- Assuming the size of the stack frame of a function is known, divide it in variables in a useful but conservative way.
- In particular, recover the type (size, signedness, etc).

## References

- `https://rev.ng`

CGEN is a subproject of GCC which aims to make completely explicit the behavior of the instruction of a certain architecture.

## Project goals

- Translate each CGEN instruction to LLVM IR.
- Use it as an alternate frontend to `rev.ng`.

## References

- `https://rev.ng`
- `https://github.com/embecosm/cgen/blob/master/cpu/cris.cpu`

Currently `rev.ng` only supports static ELF binaries.

## Project goals

- Implement the logic for loading dynamic ELF binaries; or
- Implement the logic for loading other binary image formats, in particular PE/COFF or Mach-O.

## References

- `https://rev.ng`
- `https://clearmind.me/presentations/linking.pdf`

# Static Binary Analysis with rev.ng

Currently `rev.ng` can only translate programs compiled for Linux for ARM, MIPS and x86_64.

## Project goals (choose one!)

- Add support and test functionality of one other architecture supported by QEMU, in particular x86, AArch64 (ARM 64 bits), RISC-V, OpenRISC, PPC and SPARC.

## References

- `https://rev.ng`

Currently `rev.ng` can only translate programs compiled for Linux for ARM, MIPS and x86_64.

## Project goals (choose one!)

- Complete the support for Linux: certain syscalls are not currently supported by rev.ng. In particular support for multithreading would be beneficial.
- Add support for BSD-like operating systems: QEMU's user mode can also emulate the BSD-kernel. This step consists in integrating it with rev.ng.

## References

- `https://rev.ng`