Gcc under the hood

Siddhesh Poyarekar

Linaro connect
San Diego 2019

The Next Hour

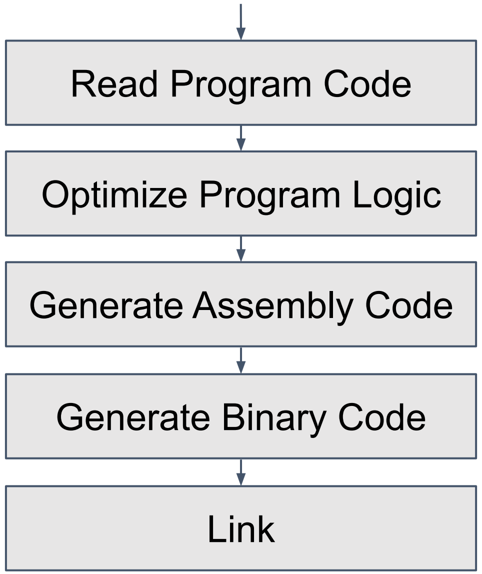
- The journey from source to binary
  - Understand how your source code goes through gcc, as and finally ld
- Digging deeper
  - We explore how intermediate code generated during compilation can help us understand our program. And the compiler.
- Hacking
  - We look at some use cases for enhancing the compiler

I...

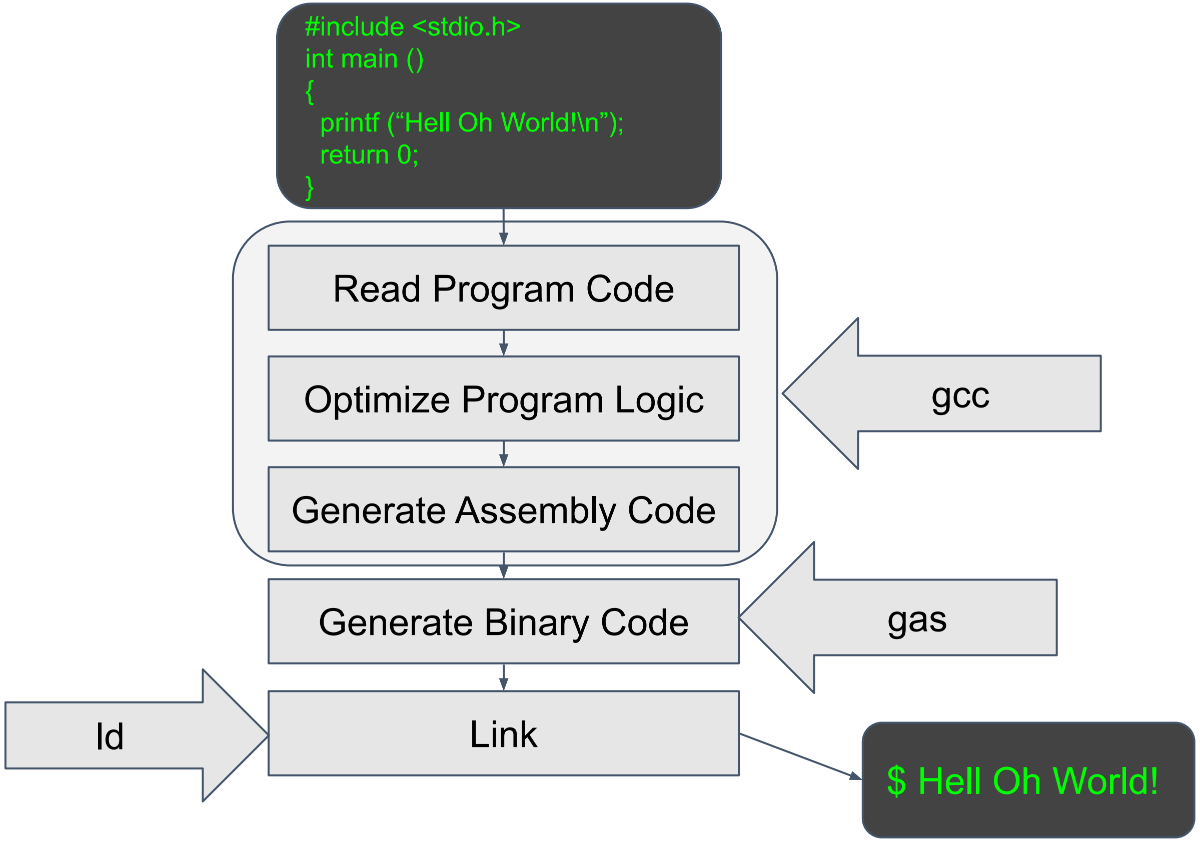
- Am Toolchain Tech Lead at Linaro Developer Services
    - We provide professional toolchain services for the Arm ecosystem
- Maintain the GNU C Library
- Contribute to GCC/binutils
- Hack on LuaJIT

From Source to Binary

The 30 second view of a toolchain

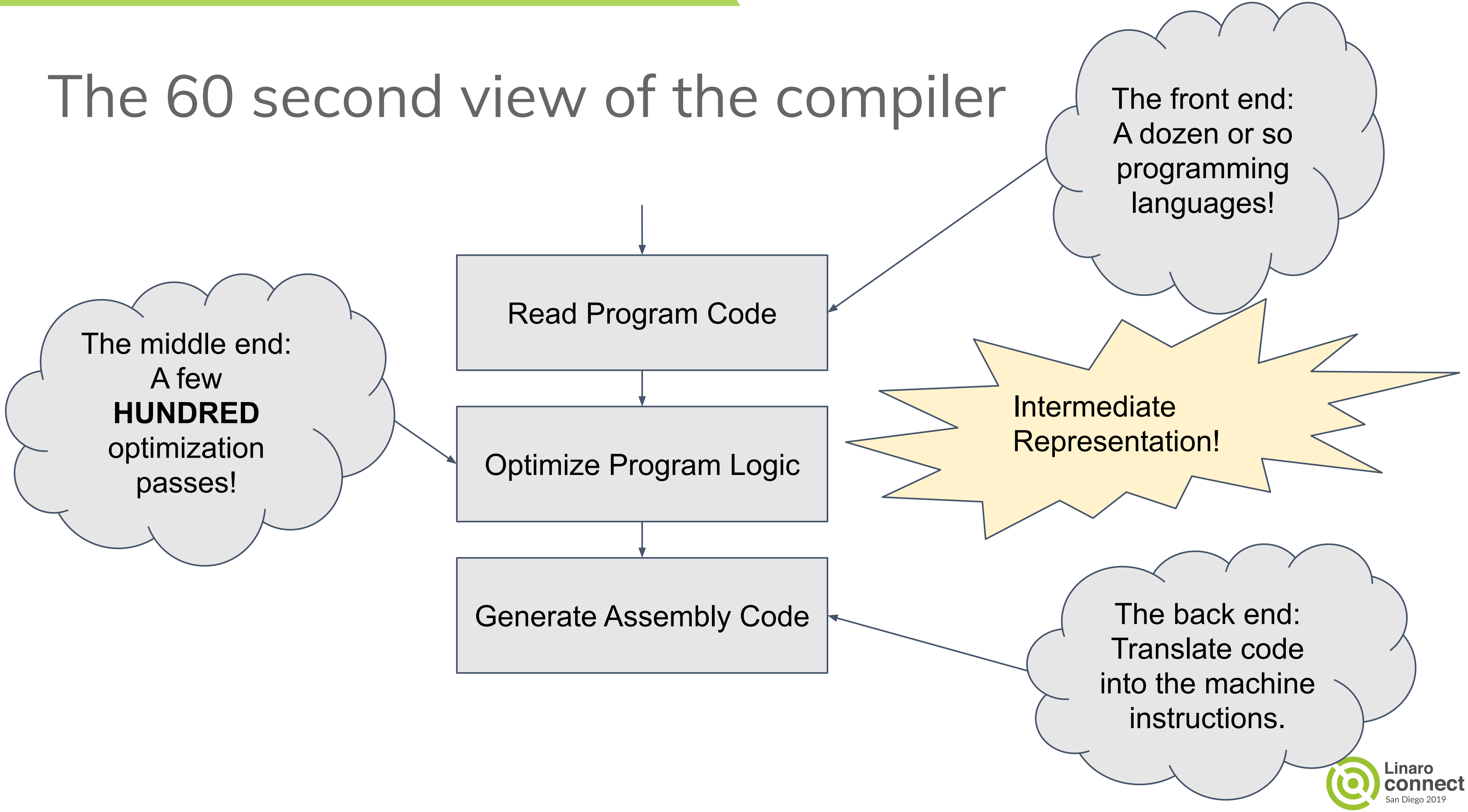


What this means for us

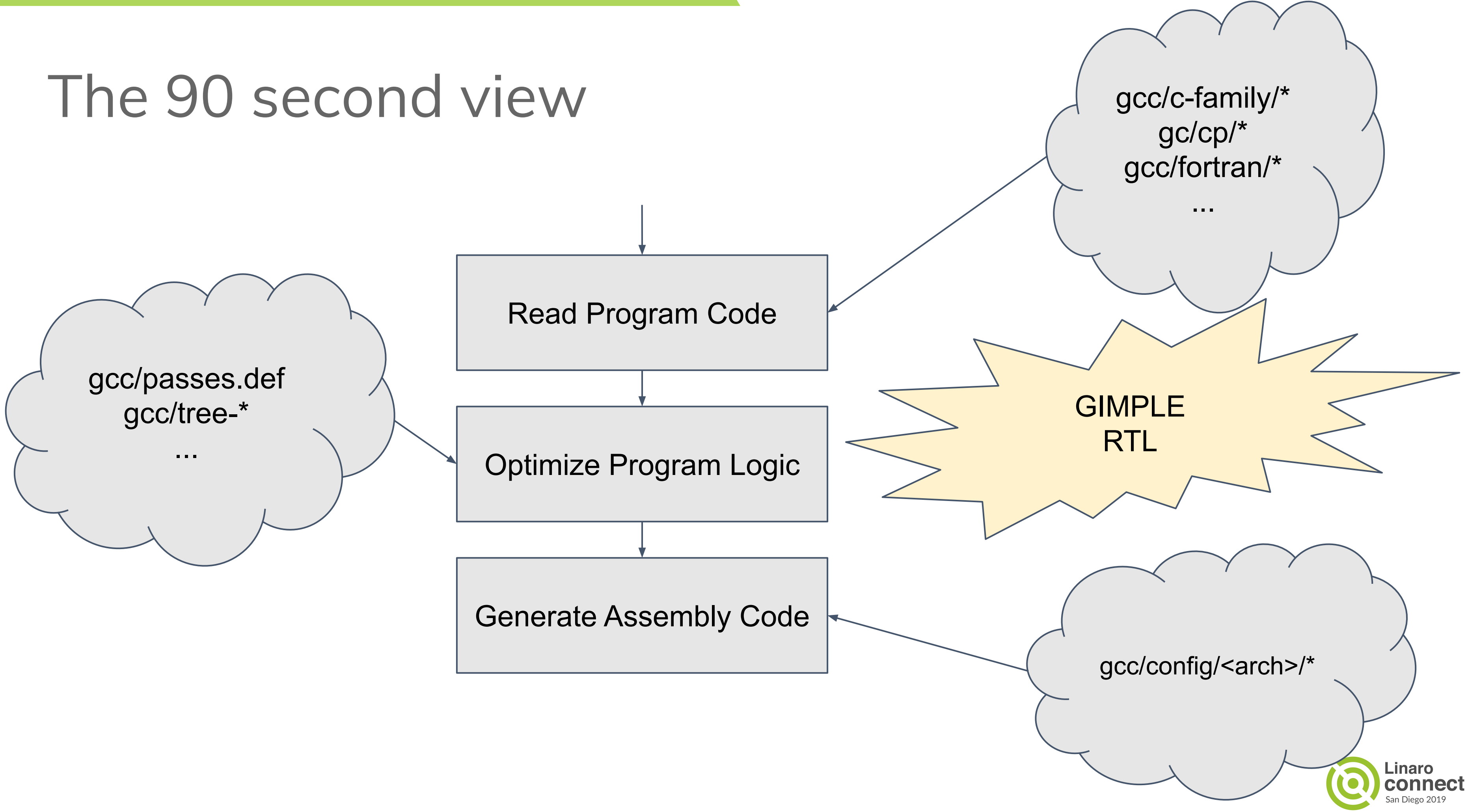

```
#include <stdio.h>
int main ()
{
  printf (“Hell Oh World!\n”);
  return 0;
}
```

```
$ Hell Oh World!
```

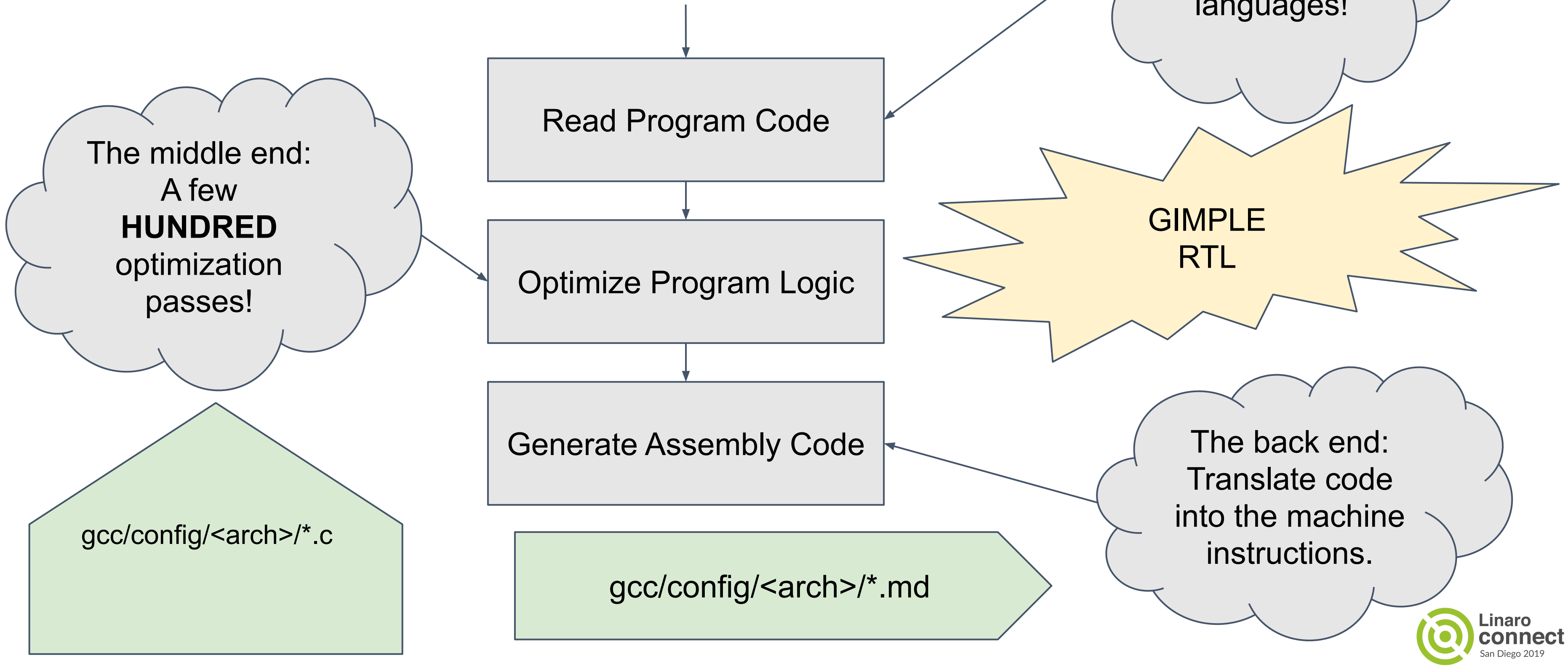
The 60 second view of the compiler



The 90 second view



The 120 second view



Digging Deeper

Intermediate Representations

- GENERIC
  - Tree structure representation of a function
  - Interface between the parser and optimiser
- GIMPLE
  - Three address, machine and language independent format
  - Lowered from GENERIC
  - More restrictive than GENERIC
- RTL
  - Lowest Intermediate representation
  - Sequential instruction descriptions lowered from GIMPLE
  - Expressed as Lisp-like S-expressions

GENERIC

```
struct GTY(())
tree_<specialization> {
  struct tree<type it is
based on>;
  <The contents of the tree>;
};

e.g.

struct GTY(()) tree_string {
  struct tree_typed typed;
  int length;
  char str[1];
};
```

- Tree structure with connected `tree_node`s (gcc/tree-core.h)
- Everything based on the tree_base struct
    - Look for struct GTY(()) tree_base

GIMPLE

- The optimizer workhorse
- Linear statements with no more than 3 operands in most cases
- Tuples defined in gcc/gimple.def
- Control flow described by the Control Flow Graph (CFG)

Control Flow Graphs

- Overlays on GIMPLE and RTL
- Graph that connects basic blocks (BB) of sequential code
    - Each BB may have one or more GIMPLE tuples
- Edges describe flow of control from one BB to another
- See gcc/cfg.* for more details
- Loops get special treatment
    - See gcc/cfgloop.h for details

GIMPLE Single Static Assignment (SSA)

- Variables are assigned in exactly one location
- Multiple assignments result in multiple copies of the variable

```
x = 10; x += 20;
```

Becomes

```
x_1 = 10; x_2 = x_1 + 20;
```

- Conditional assignments result in mysterious entities called PHI nodes

```
if (n > 10) x = 10;
else x = 20;
return x;
```

Becomes

```
if (n > 10) x_1 = 10;
else x_2 = 20;
# x_3 = PHI<x_1, x_2>;
return x_3;
```

Register Transfer Language (RTL)

- Low level representation intended to map directly to one or more instructions
- Internal structure form as well as a textual form made of Lisp-like S-expressions
- RTL expressions are listed in rtl.def
- Textual form used to write a machine description

Machine Description

- We want assembly in the end
- *.md files with RTL instruction descriptions
  - A gcc preprocessing tool parses it and generates code
- An RTL instruction may expand into one or more machine instructions
- One machine description file per architecture

Extending the Machine Description

- Additional sources per architecture to make more intelligent decisions about generated code
- Source files in config/<arch>/*.c

Optimisation Passes

- Tree Level Optimisers
  - Static data flow analysis on tree IR (GIMPLE)
  - Machine Independent
  - E.g. DCE, CSE, IV optimisation, vectorisation
- RTL Optimisers
  - Static analysis on sequential IR
  - Machine dependent
  - E.g. register allocation, instruction scheduling, etc.
- Plug Your Own Optimiser
  - Add to passes.def

Peeking and Poking

Studying Intermediate Outputs

- -fdump-tree-* options to study GIMPLE IR outputs for every pass
- -fdump-rtl-* options to study RTL IR outputs for every pass

Squeezing the last drop

- Microarchitecture descriptions
- Machine descriptions with pipeline information
- Used by the instruction scheduler pass to select or reorder instructions
- Per-cpu cost tables
  - Loop alignment
  - Function alignment
  - Costs of operations (e.g. unaligned access)

96 Boards .org
96Boards is a range of specifications with boards and peripherals offering different performance levels and features in a standard footprint.

Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org

Linaro connect
San Diego 2019