2023 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION CHINA

accellera SYSTEMS INITIATIVE

DESIGN AND VERIFICATION™ CONFERENCE AND EXHIBITION

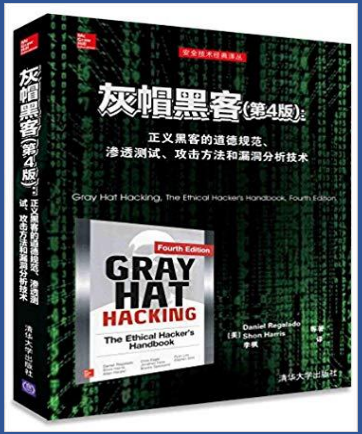
Shanghai | September 20, 2023

Python-based emerging DSLs for

FOSS EDA

Feng Li (李枫)
hkli2012@126.com
Sep 20, 2023

Who Am I



An indie developer from China

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)
- Pure software developement for ~15 years(~11 years on Mobile dev)
- Actively participating Open Source Communities:
  https://github.com/XianBeiTuoBaFeng2015/MySlides
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, IoT, Programming Languages & Runtimes, Network, Virtualization, RISC-V, EDA, 5G/6G…

Agenda

I. Background

- Technology Stack
- Testbeds

II. Practice & Exploration

- Exo
- Mojo/xDSL
- Acton
- Ray

III. Future Work

- Next generation system language
- New VMs for emerging workloads

IV. Wrap-up

I. Background
1) Technology Stack
1.1 Exo

- https://exo-lang.dev/

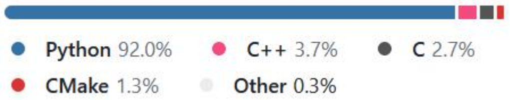
  A low-level language (and Exocompiler) designed to help performance engineers write, optimize, and target high-performance computing kernels onto new hardware accelerators.

  ## What does Exo do?

  Exo is a domain-specific programming language that helps low-level performance engineers transform very simple programs that specify what they want to compute into very complex programs that do the the same thing as the specification, only much, much faster.

  https://github.com/exo-lang/exo

  Languages



  ...

- For more details, you may refer to our previous talk "Exo--A new programming language for hardware accelerators" at OSDT 2022 and the upcoming follow-ups.

I. Background

HelloWorld

Hello Exo!

Let's write a naive matrix multiply function in Exo. Put the following code in a file called `example.py`:

```
# example.py
from __future__ import annotations
from exo import *

@proc
def example_sgemm(
    M: size,
    N: size,
    K: size,
    C: f32[M, N] @ DRAM,
    A: f32[M, K] @ DRAM,
    B: f32[K, N] @ DRAM,
):
    for i in seq(0, M):
        for j in seq(0, N):
            for k in seq(0, K):
                C[i, j] += A[i, k] * B[k, j]
```

And now we can run the exo compiler:

```
$ exocc -o out --stem example example.py
$ ls out
example.c  example.h
```

These can either be compiled into a library (static or shared) or compiled directly into your application. You will need to write a short runner program yourself to test this code. For example:

```
// main.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "example.h"

float* new_mat(int size, float value) {
  float* mat = malloc(size * sizeof(*mat));
  for (int i = 0; i < size; i++) {
    mat[i] = value;
  }
  return mat;
}

int main(int argc, char* argv[]) {
  if (argc != 4) {
    printf("Usage: %s M N K\n", argv[0]);
    return EXIT_FAILURE;
  }
```

Then this can be easily compiled and run:

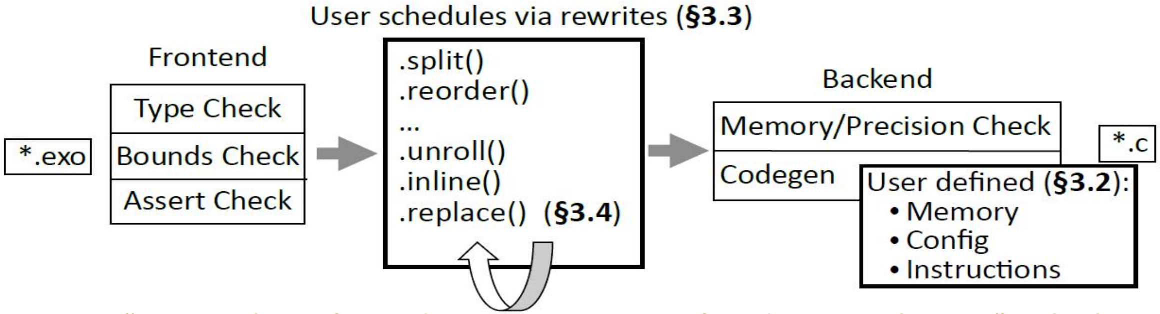
```
$ gcc -I out/ -o runner main.c out/example.c
$ ./runner 128 128 128
Each iteration ran in 11590 milliseconds
```

Source: https://exo-lang.dev/

I. Background

How it works

◆



Source: "Exocompilation for Productive Programming of Hardware Accelerators", Yuka Ikarashi et al, PLDI 2022.

1) Pioneers

Halide(https://halide-lang.org/), TVM(https://tvm.apache.org/) etc.

2) Main external dependencies

ASDL(https://github.com/ChezJrk/asdl)

a modern Python (3.8+) library for generating helpful algebraic data types out of ASDL(Abstract Syntax Definition Language) definitions.

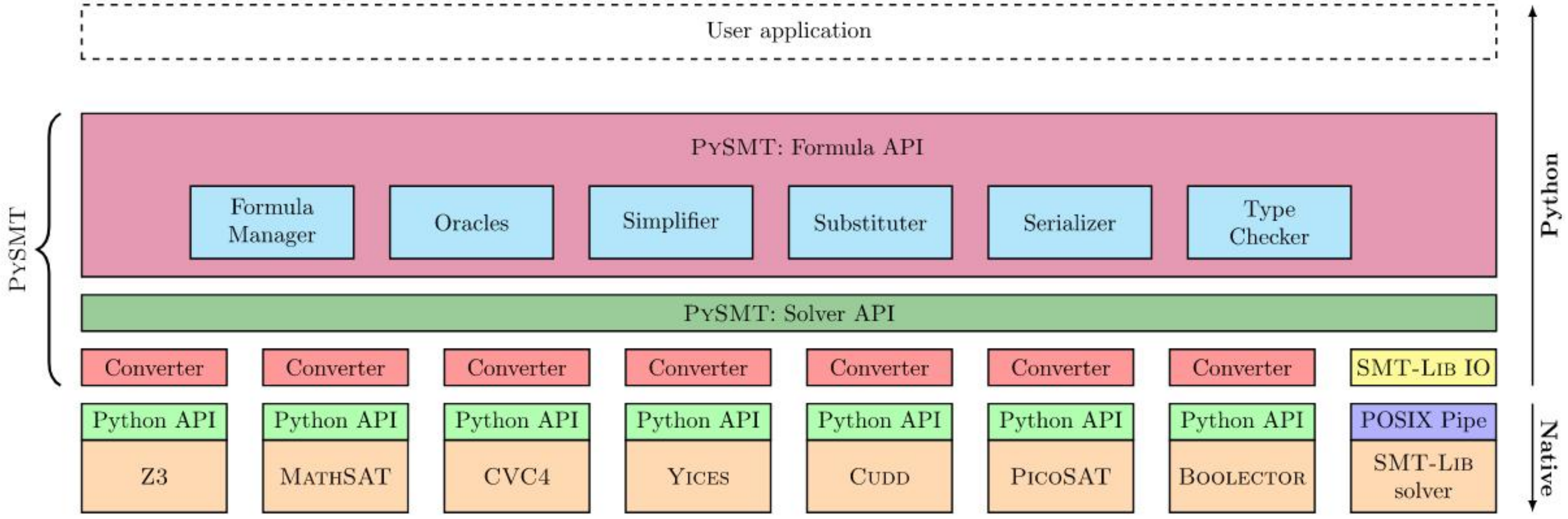
PySMT(https://github.com/pysmt/pysmt)

a Python library for SMT(Satisfiability Modulo Theory) formulae manipulation and solving)/Z3(a famous theorem prover from Microsoft Research).

I. Background

pySMT makes working with **Satisfiability Modulo Theory** simple:

- Define formulae in a *simple*, *intuitive*, and *solver independent* way
- Solve your formulae using one of the native solvers, or by wrapping any SMT-Lib compliant solver,
- Dump your problems in the SMT-Lib format,
- and more...



...

I. Background

1.2 Mojo

- https://www.modular.com/mojo

Mojo 🔥 — the programming language for all AI developers.

Mojo combines the usability of Python with the performance of C, unlocking unparalleled programmability of AI hardware and extensibility of AI models.

- https://github.com/modularml/
- https://mojolang.org/

A new Python eDSL (also designed as a superset of Python) from Modular AI (building a platform with the intent to unify the world's ML/AI infrastructure) that founded by "The farther of LLVM" (Chris Lattner).

It means a programming language with powerful compile-time metaprogramming, integration of adaptive compilation techniques, caching throughout the compilation flow, and other things that are not supported by existing languages.

From the perspective of implementation, Mojo is not only built on top of MLIR but also provides a way to access it.

Mojo may be the biggest programming language advance in decades

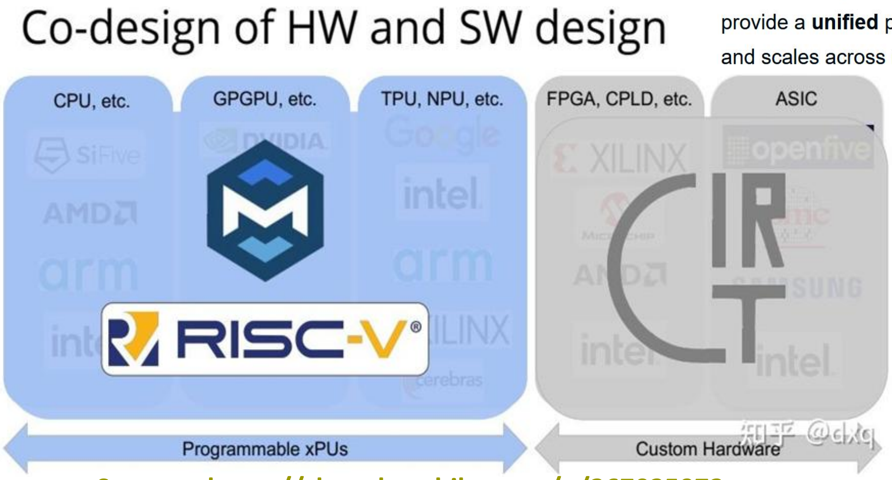
Mojo is a new programming language, based on Python, which fixes Python's performance and deployment problems.

可能是过去三十年来编程语言最大的革新：新的面向AI的编程语言Mojo发布~

Our predictions: Mojo will become one of the key technologies to AGI.

...

I. Background



Source: https://zhuanlan.zhihu.com/p/367035973

Mojo is a programming language that is as easy to use as Python but with the performance of C++ and Rust. Furthermore, Mojo provides the ability to leverage the entire Python library ecosystem.

Mojo achieves this feat by utilizing next-generation compiler technologies with integrated caching, multithreading, and cloud distribution technologies. Furthermore, Mojo's autotuning and compile-time meta-programming features allow you to write code that is portable to even the most exotic hardware.

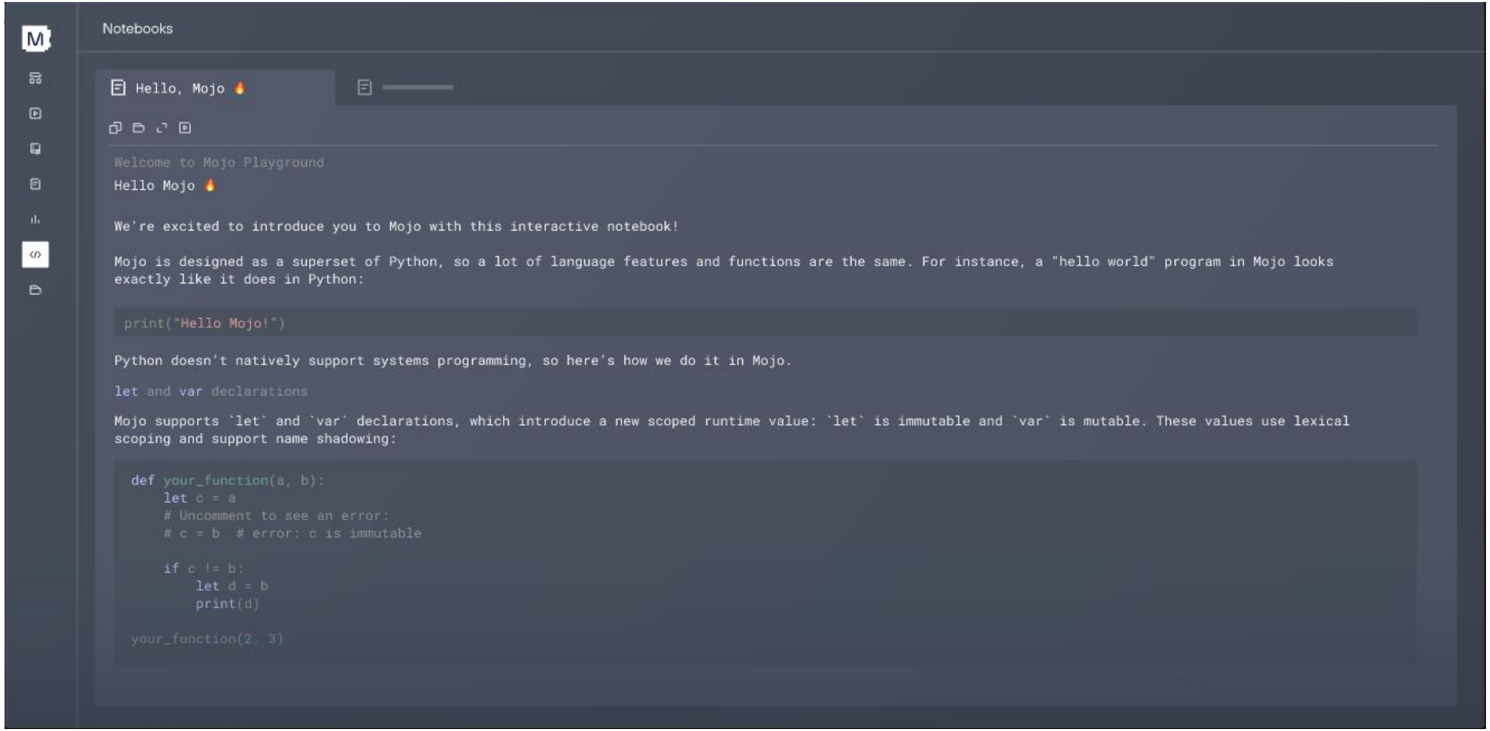
More importantly, **Mojo allows you to leverage the entire Python ecosystem** so you can continue to use tools you are familiar with. Mojo is designed to become a **superset** of Python over time by preserving Python's dynamic features while adding new primitives for systems programming. These new system programming primitives will allow Mojo developers to build high-performance libraries that currently require C, C++, Rust, CUDA, and other accelerator systems. By bringing together the best of dynamic languages and systems languages, we hope to provide a **unified** programming model that works across levels of abstraction, is friendly for novice programmers, and scales across many use cases from accelerators through to application programming and scripting.

Source: https://docs.modular.com/mojo/programming-manual.html

I. Background

HelloWorld

◆



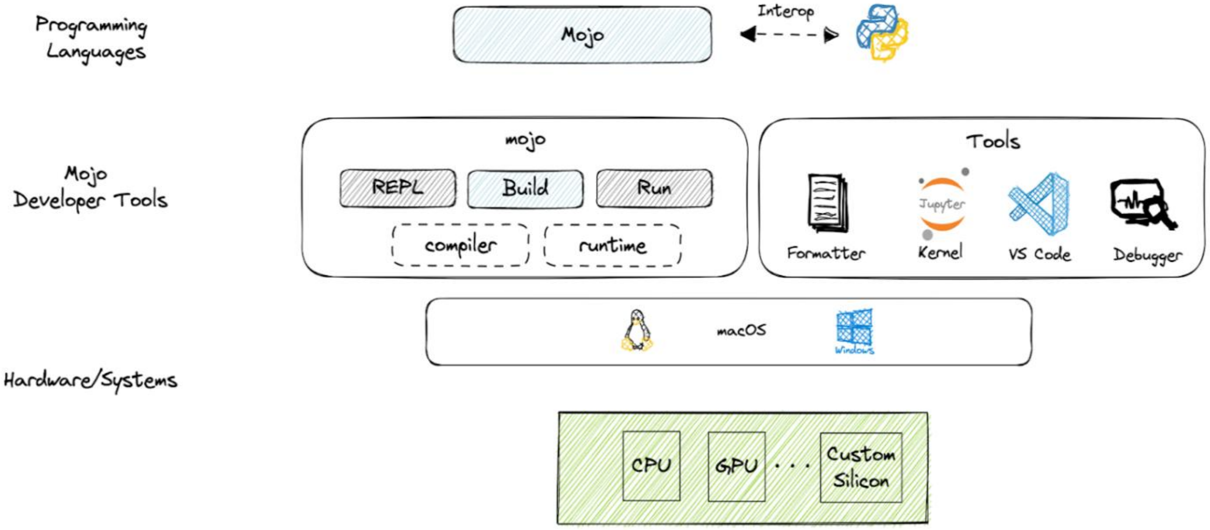
Source: https://www.modular.com/mojo

I. Background

Getting Started

- https://docs.modular.com/mojo/manual/get-started/index.html

SDK:



Source: https://www.modular.com/blog/mojo-its-finally-here

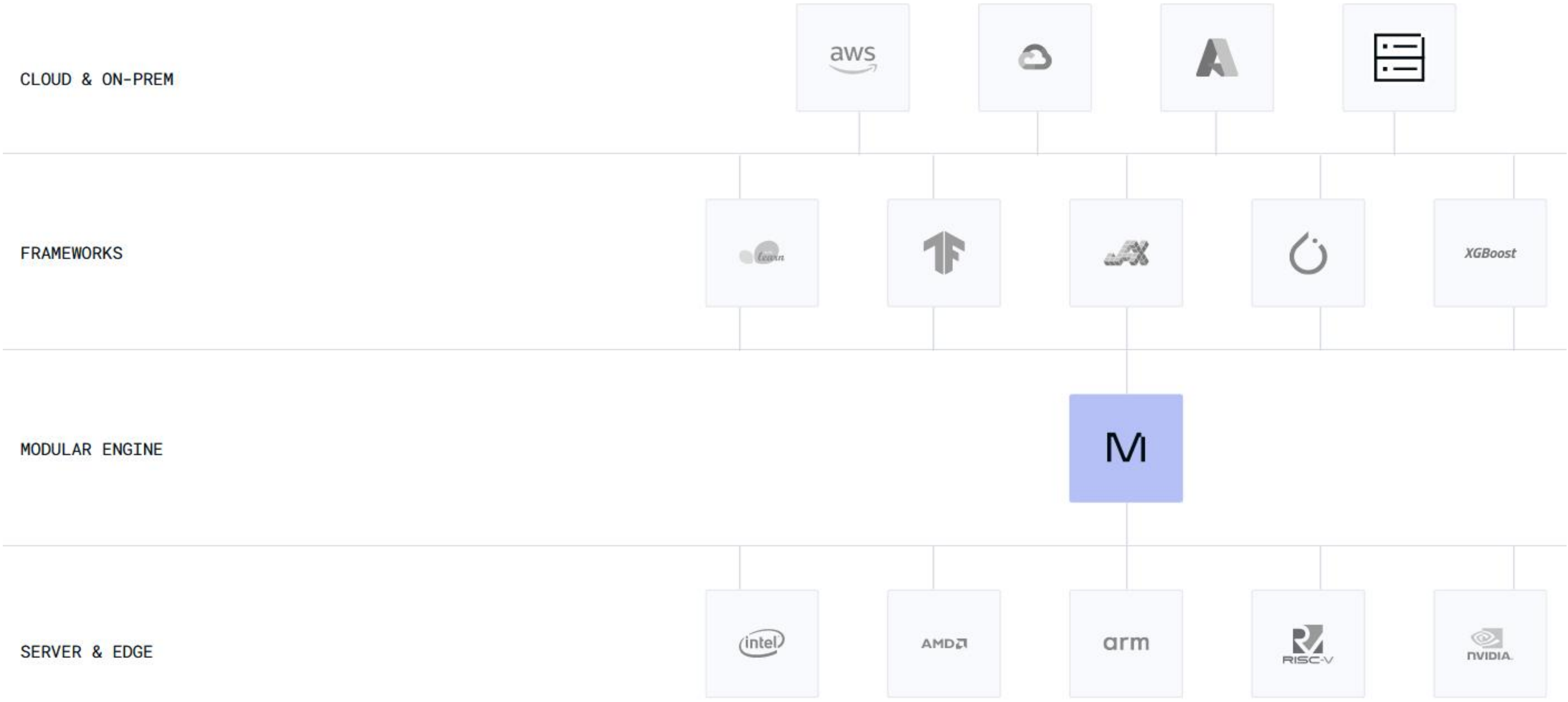
…

I. Background

Workflow of Mojo and Modular Engine

Source: https://www.modular.com/hardware

I. Background



Source: https://www.modular.com/engine

I. Background

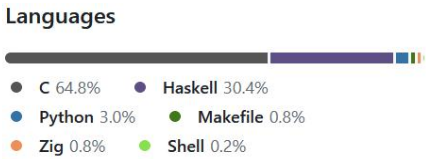
1.3 Acton

- https://www.acton-lang.org/

  A general purpose programming language, designed to be useful for a wide range of applications, from desktop applications to embedded and distributed systems by adding Actors(https://en.wikipedia.org/wiki/Actor_model) to Python, it is also a compiled language that offering the speed of C but with a considerably simpler programming model. There is no explicit memory management, instead relying on Garbage Collection. Acton is statically typed with an expressive type language and type inference.

  The Acton Run Time System (RTS) offers a distributed mode of operation allowing multiple computers to participate in running one logical Acton system. Actors can migrate between compute nodes for load sharing purposes and similar. The RTS offers exactly once delivery guarantees.

- https://github.com/actonlang/acton



I. Background

HelloWorld

- **Actors** is a key concept in Acton. Each actor is a small sequential process with its own private state. Actors communicate with each other through messages, in practice by calling methods on other actors or reading their attributes.

Source:

```
# An actor definition
actor Act(name):

    # Top level code in an actor runs when initializing an actor instance, like
    # __init__() in Python.
    print("Starting up actor " + name)

    def hello():
        # We can directly access actor arguments, like `name`
        print("Hello world from " + name)
        # TODO: remove 'return True' as it should not be necessary, but with the
        # default (returning None), we get a segfault when we do await async on
        # this method.
        return True

actor main(env):
    # Create an actor instance a of Act
    a = Act("FOO")
    # Call the actor method hello
    await async a.hello()

    await async env.exit(0)
```

Compile and run:

```
actonc actors.act
./actors
```

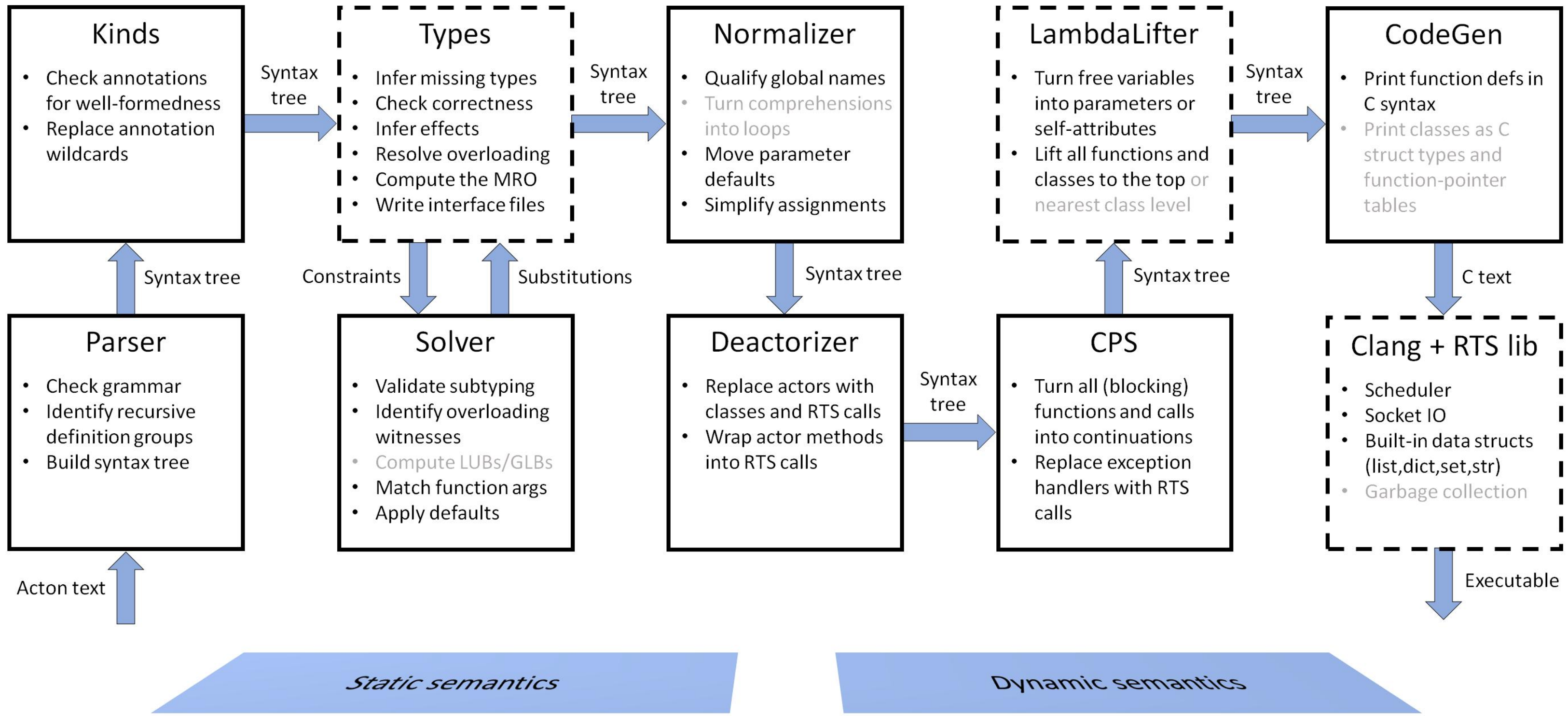
Output:

```
Starting up actor FOO
Hello world from FOO
```

Source: https://github.com/actonlang/acton

I. Background

How it works



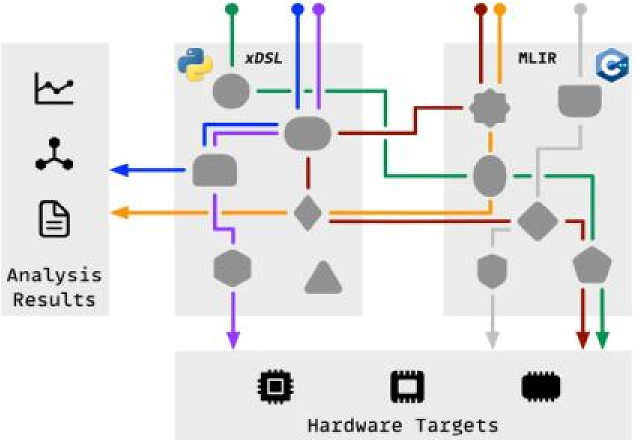
Source: https://github.com/actonlang/acton

I. Background

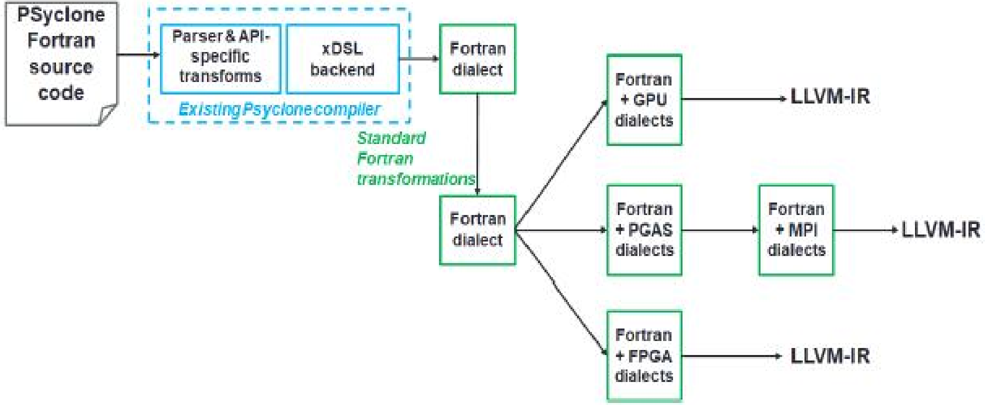
1.4 xDSL

- https://xdsl.dev/

A reimplementation of MLIR core features in pure Python which aims at bridging the Python DSL community with the MLIR one, by being fully compatible with MLIR through the textual format. Dialects can as well be translated from one framework to the other through IRDL.



Source: "xDSL: Prototyping MLIR in Python", Sasha Lopoukhine et al, European LLVM Developers' Meeting 2023.



Source: "xDSL: A common compiler ecosystem for domain specific languages", Nick Brown et al, Supercomputing Conference 2022.

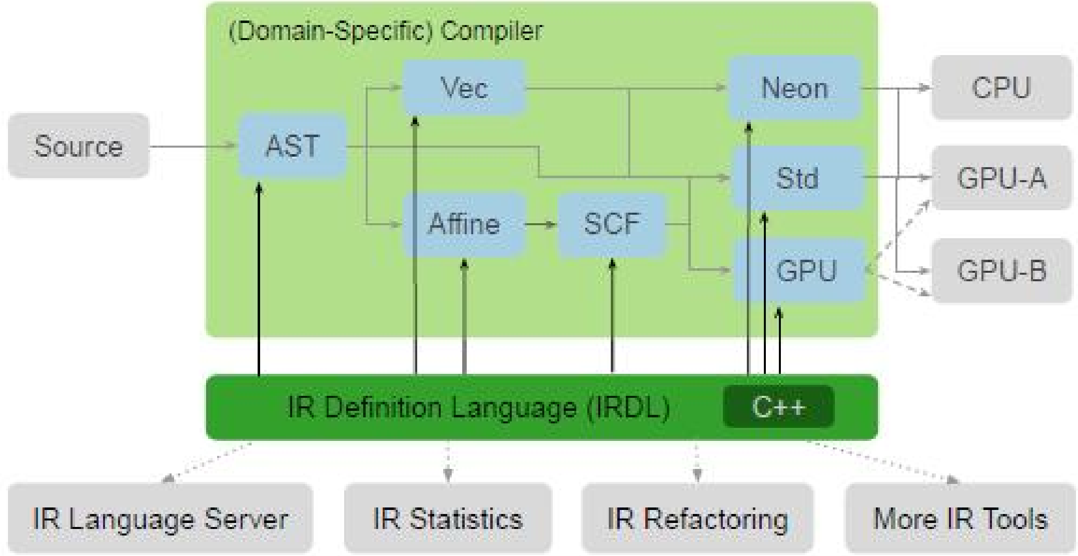
https://github.com/xdslproject/xdsl

I. Background

IRDL (IR Definition Language)

- https://doi.org/10.3929/ethz-b-000557152

  An IR definition language for SSA compilers.



Source: "IRDL: An IR Definition Language for SSA Compilers", Mathieu Fehr et al, PLDI 2022.

IRDL enables the concise specification of compiler IRs for the use within a multi-IR compilation flow.

It is expected to serve as a foundation for a future ecosystem of productivity increasing tooling around IR design.

- $SRC_XDSL/src/xdsl/irdl

I. Background

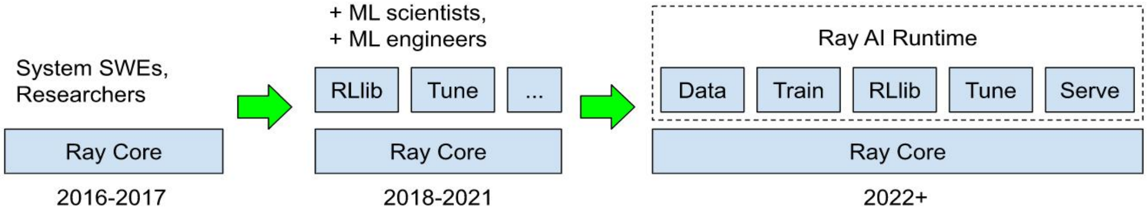
1.5 Ray

- https://www.ray.io/

  A unified framework for scaling AI and Python applications. Ray consists of a core distributed runtime and a set of AI libraries for simplifying ML compute.
- https://github.com/ray-project/ray
- https://www.anyscale.com/

Ray is the most popular open source framework for scaling and productionizing AI workloads. From **Generative AI** and **LLMs** to computer vision, Ray powers the world's most ambitious AI workloads.

History:

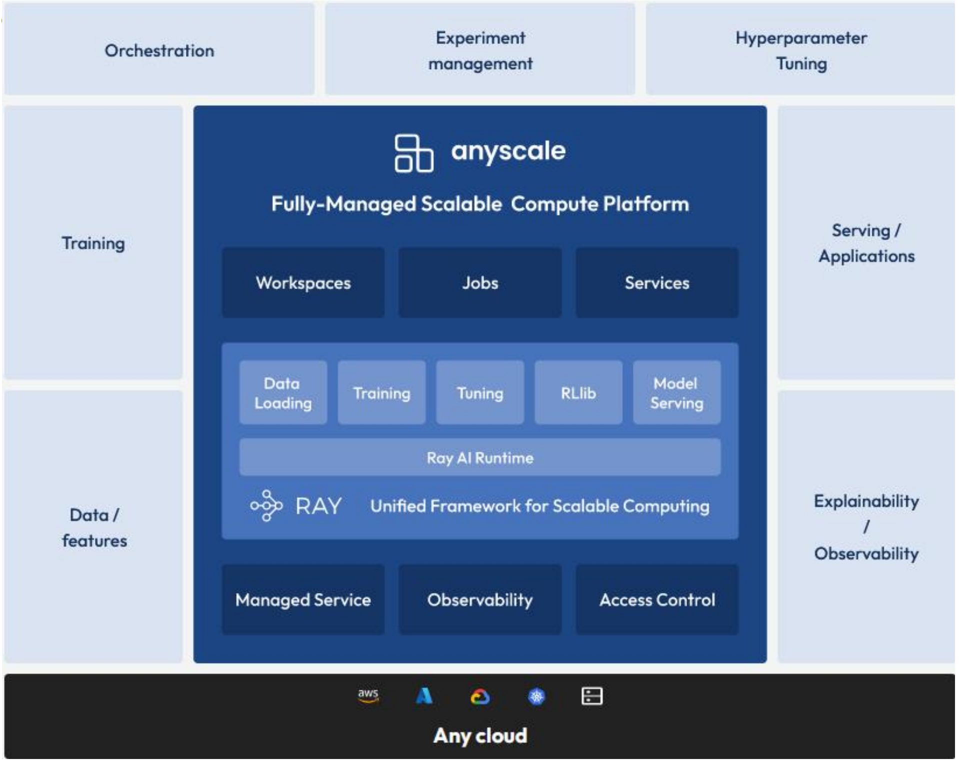


Evolution of the Ray stack and target users. AIR unifies the previously independent Ray libraries into a toolkit that works seamlessly with the ML ecosystem, enabling organizations to leverage Ray with less custom platform and integration work.

Source: Ray AIR Technical Whitepaper

...

I. Background

- 

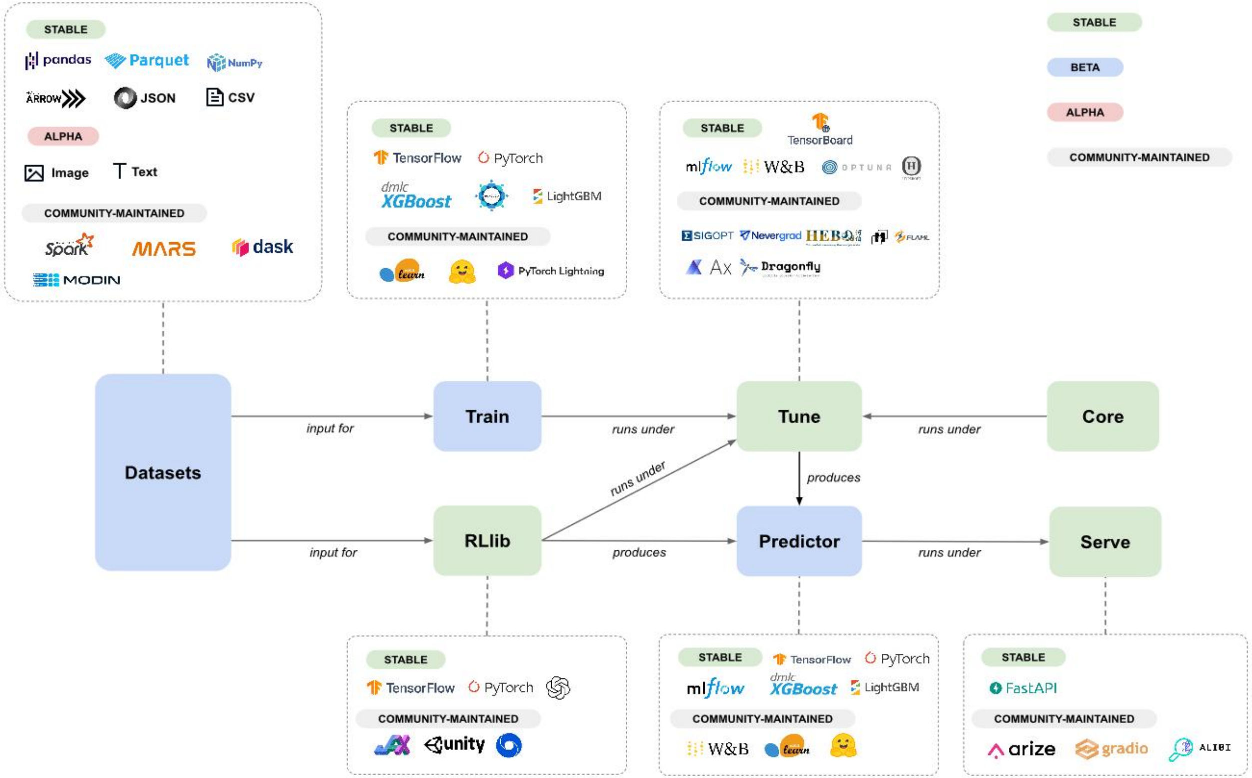
Source: https://www.anyscale.com/platform

- For more details, you may refer to our previous talk "Ray--A Swiss Army Knife for Distributed Computing & AI" at COSCon 2022 and the upcoming follow-ups.

I. Background

Ecosystem

◆



Source: https://docs.ray.io/en/master/_images/air-ecosystem.svg

I. Background

Code demo

```python
# First, decorate your function with @ray.remote to declare that you want to run this function remotely.
# Lastly, call that function with .remote() instead of calling it normally.
# This remote call yields a future, or ObjectRef that you can then fetch with ray.get.

@ray.remote
def f(x):
    return x * x

futures = [f.remote(i) for i in range(4)]
print(ray.get(futures)) # [0, 1, 4, 9]
```

```
[0, 1, 4, 9]
```

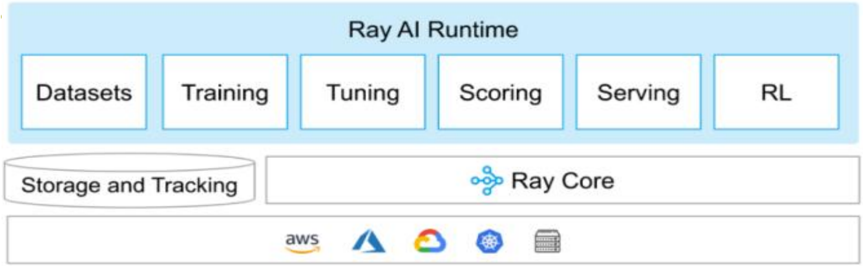
```python
# Ray provides actors to allow you to parallelize an instance of a class in Python.
# When you instantiate a class that is a Ray actor, Ray will start a remote instance of that class in the cluster.
# This actor can then execute remote method calls and maintain its own internal state.

@ray.remote
class Counter(object):
    def __init__(self):
        self.n = 0

    def increment(self):
        self.n += 1

    def read(self):
        return self.n

counters = [Counter.remote() for i in range(4)]
[c.increment.remote() for c in counters]
futures = [c.read.remote() for c in counters]
print(ray.get(futures)) # [1, 1, 1, 1]
```

```
[1, 1, 1, 1]
```

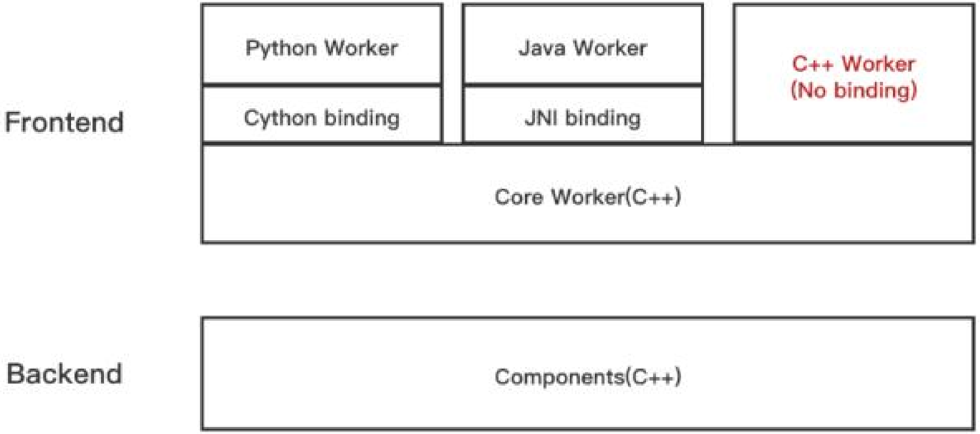
Source: https://www.databricks.com/notebooks/raydemo.html

I. Background

Overall design

- 

Source: https://www.anyscale.com/blog/announcing-ray-2-0



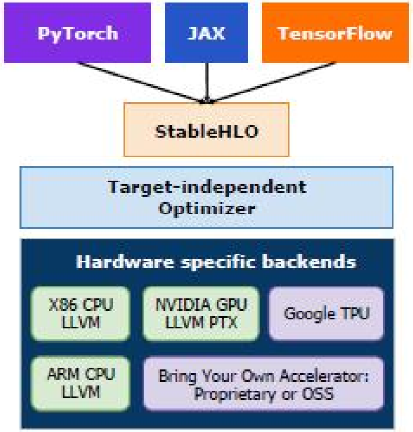
Source: https://www.anyscale.com/blog/modern-distributed-c-with-ray

I. Background

1.6 OpenXLA

- https://github.com/openxla
  An open-source ML compiler ecosystem co-developed by Alibaba, AWS, AMD, Apple, ARM, Google, Intel, Meta, NVIDIA, and more, which using the best of XLA (https://www.tensorflow.org/xla) & MLIR.
  It aims at accelerate and simplify ML development by defragmenting the ML stack across frontend frameworks and hardware backends.

OpenXLA



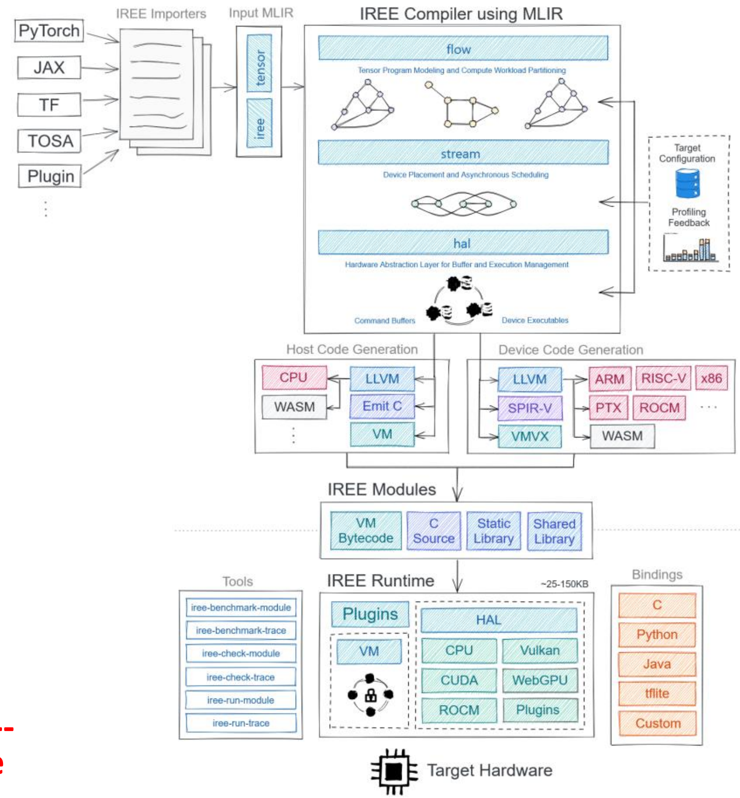
Source: https://pytorch.s3.amazonaws.com/posters/ptc2022/H01.pdf

I. Background

IREE (Intermediate Representation Execution Environment)

- https://openxla.github.io/iree/
  An MLIR-based end-to-end compiler and runtime that lowers ML models to a unified IR that scales up to meet the needs of the datacenter and down to satisfy the constraints and special considerations of mobile and edge deployments.
  It adopts a holistic approach towards ML model compilation: the IR produced contains both the scheduling logic, required to communicate data dependencies to low-level parallel pipelined hardware/API like Vulkan, and the execution logic, encoding dense computation on the hardware in the form of hardware/API-specific binaries like SPIR-V(the industry open standard intermediate language for parallel compute and graphics).

- For more details, you may refer to our previous talk "IREE--MLIR-based end-to-end compiler and runtime for Machine Learning" at OSDT 2022 and the upcoming follow-ups.



Source: https://openxla.github.io/iree/#project-architecture

I. Background

1.7 The others

- For the rest of technology stack that needed by this topic, please refer to corresponding part of the topic "Beyond UVM" of mine @DVCon China 2023.

I. Background

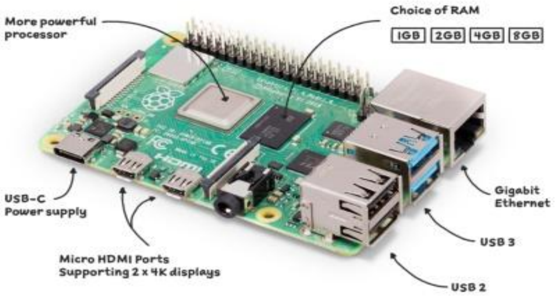
2) Testbeds

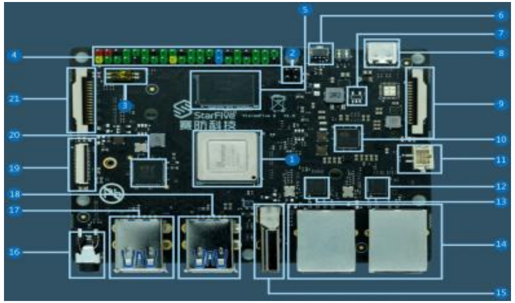
- **HW/SW**

  Testbed1: Intel NUC X15 LAPAC71H(32GB DDR5) with Fedora 38(Linux Kernel 6.3.11/6.4.15)

  Testbed2: Raspberry Pi 4 (8GB LPDDR4) with Fedora 37(Linux Kernel 6.3.8/6.4.12);

  Testbed3: VisionFive 2(8GB LPDDR4) with Debian 12(Linux Kernel 5.15).





II. Practice & Exploration

1) Exo

Install and test from the source directly:

pip install -e . --verbose --user

```
[mydev@fedora exo-master]$ pip install -e . --verbose --user
...
  running build_py

          Editable install will be performed using .pth file to extend `sys.path` with:
          ['src']

  Options like `package-data`, `include/exclude-package-data` or
  `packages.find.exclude/include` may have no effect.

  adding '__editable__.exo_lang-0.0.2.pth'
  creating '/tmp/pip-wheel-r2k44w6j/.tmp-5w9b4v71/exo_lang-0.0.2-0.editable-py3-none-any.whl' and adding '/tmp/tmp3l04p59jexo_lang-0.0.2-0.editable-py3-none-any.whl' to it
  adding 'exo_lang-0.0.2.dist-info/LICENSE.md'
  adding 'exo_lang-0.0.2.dist-info/METADATA'
  adding 'exo_lang-0.0.2.dist-info/WHEEL'
  adding 'exo_lang-0.0.2.dist-info/entry_points.txt'
  adding 'exo_lang-0.0.2.dist-info/top_level.txt'
  adding 'exo_lang-0.0.2.dist-info/RECORD'
  Building editable for exo-lang (pyproject.toml): finished with status 'done'
  Created wheel for exo-lang: filename=exo_lang-0.0.2-0.editable-py3-none-any.whl size=9048 sha256=65770591eb45abe237a496af15c9f3dbf4d59476a9b114237b769ed67de58d27
  Stored in directory: /tmp/pip-ephem-wheel-cache-9ac5qkds/wheels/2e/ad/c1/55e73f4ac45ceb5b690300141cc25a82bc68c8224980bf3baf
Successfully built exo-lang
Installing collected packages: exo-lang
  changing mode of /home/mydev/.local/bin/exocc to 755
Successfully installed exo-lang-0.0.2

 ...
```

```
[mydev@fedora exo-master]$ pytest
============================= test session starts ==============================
platform linux -- Python 3.11.4, pytest-7.1.3, pluggy-1.0.0
rootdir: /opt/MyWorkSpace/MyProjs/HW-EDA/Languages/Novel/Exo/Official/exo-master, configfile: pyproject.toml
plugins: anyio-3.6.2, cocotb-test-0.2.4, forked-1.4.0, cov-4.0.0, xdist-2.5.0, sugar-0.9.6
collected 486 items / 1 skipped

tests/test_apps.py ...
..                                                   [  1%]
tests/test_codegen.py ....................                                  [  5%]
tests/test_config.py ss...sss..ss.....                                      [  8%]
tests/test_cursors.py ..................                                    [ 12%]
tests/test_effects.py ....ss..................................              [ 20%]
tests/test_error_reporting.py s                                             [ 20%]
tests/test_im2col.py .                                                      [ 20%]
tests/test_internal_cursors.py ........................................     [ 29%]
....                                                                        [ 29%]
tests/test_interpreter.py ...                                               [ 30%]
tests/test_neon.py .EEEE.E                                                  [ 31%]
tests/test_new_eff.py ..................                                    [ 35%]
tests/test_precision.py ....                                                [ 36%]
tests/test_range_analysis.py ..............                                 [ 39%]
tests/test_reflection.py .......                                            [ 40%]
tests/test_rvv.py E                                                         [ 40%]
tests/test_schedules.py ..................s.............................    [ 50%]
.............................................s..........................    [ 65%]
......................................................................      [ 79%]
tests/test_typecheck.py ...............................s.............       [ 88%]
tests/test_uast.py ...                                                      [ 89%]
tests/test_window.py ........                                               [ 91%]
tests/test_x86.py EE.E..EE.EEEEEEEEEEEEEEEEEE                               [ 96%]
tests/amx/test_amx_instr.py ssssss.sssss                                    [ 99%]
tests/pldi22/test_gemmini_conv_ae.py .                                      [ 99%]
tests/pldi22/test_gemmini_matmul_ae.py .                                    [ 99%]
tests/pldi22/test_gemmini_matmul_paper.py .                                 [100%]

==================================== ERRORS ====================================
_____________________ ERROR at setup of test_neon_memcpy _______________________

 ...
===== 432 passed, 25 skipped, 4 warnings, 30 errors in 1784.47s (0:29:44) ======
```

For Install and Test Exo on master branch with last commit 700fe3bb00ab9564eadf35787cba60edd02e92c0) on **Testbed2**

II. Practice & Exploration

2) Mojo/xDSL

2.1 Mojo

2.1.1 llama2.mojo

- https://github.com/tairov/llama2.mojo

Inference Llama 2 in one file of pure 🔥

why this port?

This repository serves as a port that provides a Mojo-based implementation of `llama2.c`.

With the release of Mojo, I was inspired to take my Python port of llama2.py and transition it to Mojo. The result? A version that leverages Mojo's SIMD & vectorization primitives, boosting the Python performance by nearly 250x. Impressively, the Mojo version now outperforms the original `llama2.c` compiled in `runfast` mode out of the box by 15-20%. This showcases the potential of hardware-level optimizations through Mojo's advanced features. I think this also can help us to see how far can we go with the original `llama2.c` hardware optimizations.

II. Practice & Exploration

Performance

- Since there were some debates was this comparison legit or not I did some research and found that in `runfast` mode `llama2.c` includes multiple optimizations like aggressive vectorization, which makes comparison fair with Mojo SIMD vectorization.

  Further researches of both solutions in parallelized mode compilation showed that `llama2.c` is faster by ~20% I'm still investigating in this direction since not all the possible optimizations were applied to the Mojo version so far.

benchmarking

OS/HW specs

```
OS:         Ubuntu 20.04
CPU(s):     6
Model name: Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz
CPU MHz:    3191.998
```

Model	llama2.py	llama2.c	llama2.c (runfast)	llama2.c (OMP/parallelized)	llama2.mojo	llama2.mojo (parallelized)	llama2.mojo (naive matmul)
stories15M.bin	1.3 tok/s	75.73 tok/s	237 tok/s	450 tok/s	260 tok/s	390 tok/s	67.26 tok/s
stories110M.bin	-	9 tok/s	30 tok/s	64 tok/s	40 tok/s	57 tok/s	9.20 tok/s

II. Practice & Exploration

2.2 xDSL

Install and test from the source directly:

pip install -e . --verbose --user

```
...
  Building editable for xdsl (pyproject.toml): finished with status 'done'
  Created wheel for xdsl: filename=xdsl-0.13.0+276.g0f9f42e0-0.editable-py3-none-any.whl size=10062 sha256=cacf74bd2db3
5f3b158f4ab62748a5be7118bcf0998459840946eb70b15b0c15
  Stored in directory: /tmp/pip-ephem-wheel-cache-mooybinr/wheels/9c/56/73/410d1a3af2f1fade287075d716904f5a8a05490cfd65
f1e57d
Successfully built xdsl
Installing collected packages: typing-extensions, xdsl
  Attempting uninstall: typing-extensions
    Found existing installation: typing_extensions 4.4.0
    Uninstalling typing_extensions-4.4.0:
      Removing file or directory /home/mydev/.local/lib/python3.11/site-packages/__pycache__/typing_extensions.cpython-
311.pyc
      Removing file or directory /home/mydev/.local/lib/python3.11/site-packages/typing_extensions-4.4.0.dist-info/
      Removing file or directory /home/mydev/.local/lib/python3.11/site-packages/typing_extensions.py
      Successfully uninstalled typing_extensions-4.4.0
  changing mode of /home/mydev/.local/bin/irdl-to-pyrdl to 755
  changing mode of /home/mydev/.local/bin/xdsl-opt to 755
Successfully installed typing-extensions-4.7.1 xdsl-0.13.0+276.g0f9f42e0
...
======================== 1544 passed, 1 skipped in 13.46s ========================
lit -v docs/Toy/examples --order=smart
-- Testing: 9 tests, 4 workers --
PASS: Toy :: ast.toy (1 of 9)
PASS: Toy :: scalar.toy (2 of 9)
PASS: Toy :: codegen.toy (3 of 9)
PASS: Toy :: tests/infer_shapes.mlir (4 of 9)
PASS: Toy :: tests/inline.mlir (5 of 9)
UNSUPPORTED: Toy :: tests/with-mlir/interpret.toy (6 of 9)
PASS: Toy :: tests/accelerate_toy.mlir (7 of 9)
PASS: Toy :: tests/optimise_toy.mlir (8 of 9)
PASS: Toy :: interpret.toy (9 of 9)

Testing Time: 7.88s
  Unsupported: 1
  Passed     : 8
pytest docs/Toy/toy/tests
...
PASS: xDSL :: transforms/reconcile_unrealized_casts.mlir (148 of 153)
UNSUPPORTED: xDSL :: with-riscemu/riscv_emulation.mlir (149 of 153)
PASS: xDSL :: transforms/convert-stencil-to-ll-mlir.mlir (150 of 153)
PASS: xDSL :: transforms/stencil-shape-inference.mlir (151 of 153)
PASS: xDSL :: xdsl_opt/split_input.mlir (152 of 153)
PASS: xDSL :: transforms/stencil-storage-materialization.mlir (153 of 153)

Testing Time: 93.80s
  Unsupported:  30
  Passed     : 123
...
```

For install and Test xDSL on master branch with last commit 700fe3bb00ab9564eadf35787cba60edd02e92c0) on **Testbed2**

II. Practice & Exploration

3) Acton

Install and test from the source directly:

make -j$(nproc) & make test

```
...
 Final compilation step
/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/dist/zig/zig build  --cache-dir /opt/MyWorkSpace/MyProjs/L
anguages/Acton/Official/acton-main/base/build-cache --global-cache-dir /home/mydev/.cache/acton/build-cache --prefix /o
pt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/base/out/rel --prefix-exe-dir 'bin' --prefix /opt/MyWorkSpac
e/MyProjs/Languages/Acton/Official/acton-main/base/out/rel --prefix-exe-dir 'bin' -Dtarget=aarch64-linux-gnu.2.27 -Dopt
imize=ReleaseFast -Dprojpath=/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/base -Dprojpath_outtypes=/opt
/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/base/out/types -Dsyspath=/opt/MyWorkSpace/MyProjs/Languages/Ac
ton/Official/acton-main/dist -Dsyspath_backend=../../../../../../../../../opt/MyWorkSpace/MyProjs/Languages/Acton/Offic
ial/acton-main/dist/backend -Dsyspath_base=../../../../../../../../../opt/MyWorkSpace/MyProjs/Languages/Acton/Official/
acton-main/dist/base -Dsyspath_include=/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/dist/depsout/includ
e -Dsyspath_lib=/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/dist/depsout/lib -Dsyspath_libreldev=/opt/
MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main/dist/lib/rel -Duse_prebuilt
  Finished final compilation step in  39.729 s
cp base/out/rel/lib/libActon.a dist/lib/rel/libActon.a
make dist/deps/libargp dist/deps/libbsdnt dist/deps/libgc dist/deps/libnetstring dist/deps/libprotobuf_c dist/deps/libu
tf8proc dist/deps/libuuid dist/deps/libuv dist/deps/libxml2 dist/deps/libyyjson dist/deps/pcre2
make[2]: Entering directory '/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main'
make[2]: 'dist/deps/libargp' is up to date.
make[2]: 'dist/deps/libbsdnt' is up to date.
make[2]: 'dist/deps/libgc' is up to date.
make[2]: 'dist/deps/libnetstring' is up to date.
make[2]: 'dist/deps/libprotobuf_c' is up to date.
make[2]: 'dist/deps/libutf8proc' is up to date.
make[2]: 'dist/deps/libuuid' is up to date.
make[2]: 'dist/deps/libuv' is up to date.
make[2]: 'dist/deps/libxml2' is up to date.
make[2]: 'dist/deps/libyyjson' is up to date.
make[2]: 'dist/deps/pcre2' is up to date.
make[2]: Leaving directory '/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main'
make[1]: Leaving directory '/opt/MyWorkSpace/MyProjs/Languages/Acton/Official/acton-main'
...
cd compiler && stack test
acton> configure (exe + test)
Configuring acton-0.16.0.20230729.10.16.15...
/home/mydev/.stack/programs/aarch64-linux/ghc-tinfo6-8.10.7/lib/ghc-8.10.7/bin/ghc: /lib64/libtinfo.so.6: no version in
formation available (required by /home/mydev/.stack/programs/aarch64-linux/ghc-tinfo6-8.10.7/lib/ghc-8.10.7/bin/../hask
eline-0.8.2/libHShaskeline-0.8.2-ghc8.10.7.so)
...
    test_process_write:                                              OK (8.51s)
  stdlib
    time:                                                            OK (5.39s)

4 out of 123 tests failed (494.82s)

acton> Test suite test_actonc failed
Completed 2 action(s).
Test suite failure for package acton-0.16.0.20230729.10.16.15
    test_actonc:  exited with: ExitFailure 1
Logs printed to console

make: *** [Makefile:349: test] Error 1
...
```

Patching for AArch64 as Acton officially only support X64:

```
[mydev@fedora acton-main]$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Makefile
        modified:   compiler/Acton/CommandLineParser.hs
```

For Install and Test Acton on main branch with last commit 991188fb72fd441028ebd44d5980b77f94361d2f) on Testbed2.

II. Practice & Exploration

…

```
mkdir -p deps-download
curl -o deps-download/zig-linux-x86_64-0.11.0.tar.xz https://ziglang.org/download/0.11.0/zig-linux-x86_64-0.11.0.tar.xz
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
^M  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0^M  0 42.8M    0 97011    0     0   346
k      0  0:02:06 --:--:--  0:02:06  345k^M 44 42.8M   44 19.2M    0     0  15.0M      0  0:00:02  0:00:01  0:00:01 15.
0M^M 87 42.8M   87 37.5M    0     0  16.0M      0  0:00:02  0:00:02 --:--:-- 16.0M^M100 42.8M  100 42.8M    0     0  15
.8M      0  0:00:02  0:00:02 --:--:-- 15.8M
mkdir -p dist/zig
cd dist/zig && tar Jx --strip-components=1 -f ../../deps-download/zig-linux-x86_64-0.11.0.tar.xz
rm -rf dist/zig/doc dist/zig/lib/libcxx dist/zig/lib/libcxxabi dist/zig/lib/libc/include/any-windows-any
cp -a deps/zig-extras/* dist/zig
make[1]: Leaving directory '/opt/MyWorkSpace/MyProjs/Languages/Python/DSLs/Acton/Official/acton-main'
make distribution
```

…

```
cd compiler && unset CC && unset CFLAGS && stack build --dry-run 2>&1 | grep "Nothing to build" || \
        (sed 's,^version:.*,version:      "0.16.0.20230917.19.15.6",' < package.yaml.in > package.yaml \
        && stack build --ghc-options='-j4 ' \
        && stack --local-bin-path=. install 2>/dev/null)
Preparing to install GHC (tinfo6) to an isolated location. This will not interfere with any
system-level installation.
Preparing to download ghc-tinfo6-8.10.7 ...
ghc-tinfo6-8.10.7: download has begun
ghc-tinfo6-8.10.7:  289.64 KiB / 207.64 MiB (  0.14%) downloaded...
ghc-tinfo6-8.10.7:    1.23 MiB / 207.64 MiB (  0.59%) downloaded...
mv /opt/MyWorkSpace/MyProjs/Languages/Python/DSLs/Acton/Official/acton-main/dist/depsout/bin/actondb dist/bin/actondb
rmdir /opt/MyWorkSpace/MyProjs/Languages/Python/DSLs/Acton/Official/acton-main/dist/depsout/bin
ghc-tinfo6-8.10.7:    1.40 MiB / 207.64 MiB (  0.67%) downloaded...
ghc-tinfo6-8.10.7:    1.70 MiB / 207.64 MiB (  0.82%) downloaded...
```

…

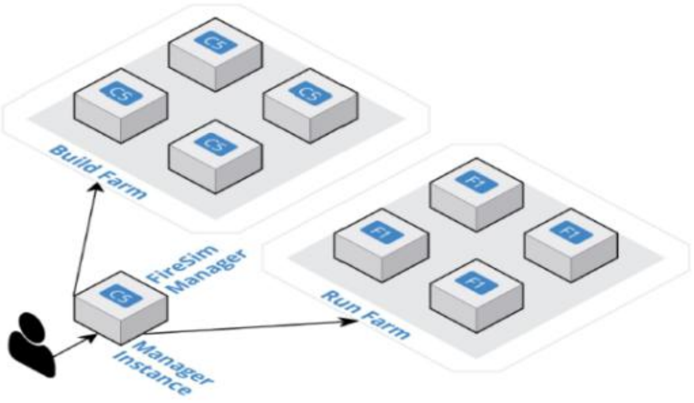
```
    test_acton_rts_sleep:                                                          OK (0.68s)
    test_net:                                                                      OK (0.78s)
    test_net_tcp:                                                                  OK (1.43s)
    test_logging:                                                                  OK (1.81s)
  stdlib
    time:                                                                          OK (0.45s)

1 out of 128 tests failed (35.00s)
```

For Install and Test Acton on main branch with last commit 144a3820ec0c028a74e25b645f30edca2904ebcd) on **Testbed1**.

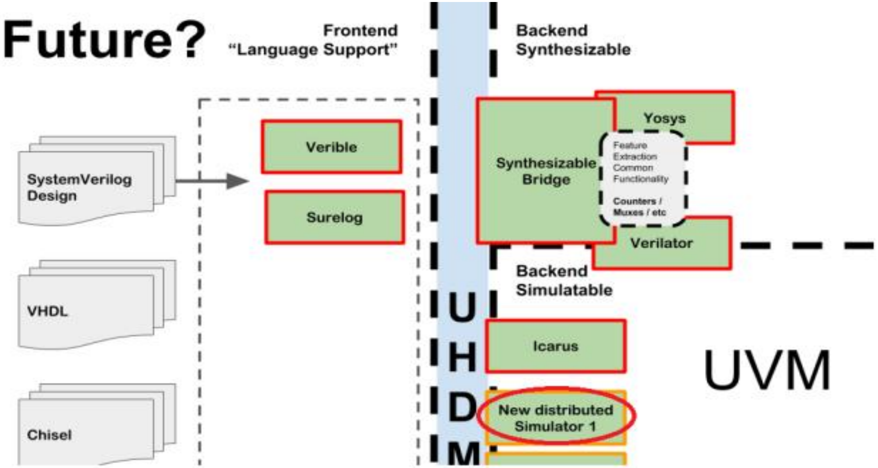
II. Practice & Exploration

4) Ray

4.1 Distributed simulation and verification

- 

Source: "Using On-Premise FPGAs and Distributed Metasimulation", Abraham Gonzalez, ISCA 2022.



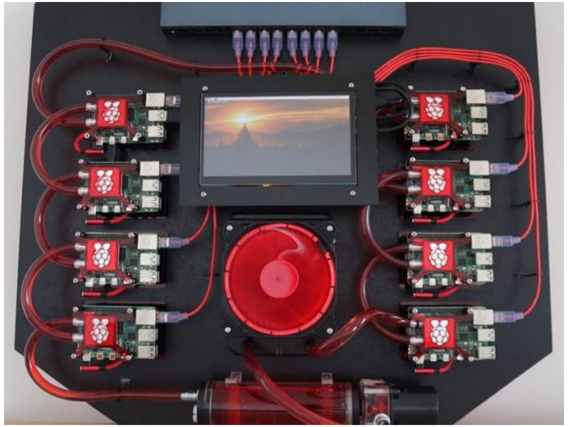
Source: https://github.com/chipsalliance/UHDM/blob/master/images/UHDM_future.png

- **Trying to use Acton to reconstruct Cocotb for parallelly running the Python testbenches across in a distributed cluster system with various RTL simulators, or even make an attempt to re-implement a distributed RTL simulator by Acton. By the way, distributed simulation and verification are the future trends in HW verification as we can observe:**

II. Practice & Exploration

Clustering at Edge

Method 1:



Method 2:

Source: https://turingpi.com/product/turing-pi-2/

II. Practice & Exploration

4.2 Beyond Kubernetes

- Now Kubernetes is the de facto standard for today's production-grade Container orchestration, which is surrounded by an amazing huge and continuously growing ecosystem. It comes with the Container first design philosophy and is optimized for that.
- Kubernetes is really powerful, but it is getting more complex and accumulating more and more historical baggage at the same time…
- Lightweight Kubernetes distribution like K3S is suitable for Edge devices that have limited computing resources when compared with Cloud side, but may not still be a good fit for hardware platform with even less computing resources like Microcontrollers which is common on IoT devices.
- New workload like Wasm and eBPF is more lightweight than Container, and their ecosystems are booming.
- Current solutions like Krustlet or Kata Containers/WasmEdge supports the above new workload by extending Kubernetes or is implemented as OCI-compatible:
  - the benefits are it can both support the Container workload and new workload, and make best of use the existed code and ecosystem of Kubernetes, Docker, and so on.
  - the drawbacks are thus means it has to inherit some kind of "historical burden", and most of all, it is not burn for new workload like Wasm and eBPF.
- The IREE runtime for AI workload also prompt us rethinking a lightweight orchestration system that specific to Bytecode VM based workload.
- …

Source: "Ray - A Swiss Army Knife for Distributed Computing & AI", Feng Li, COSCon 2022.

You may refer to our upcoming follow-ups like "First exploration of beyonding Kubernetes" etc.

II. Practice & Exploration

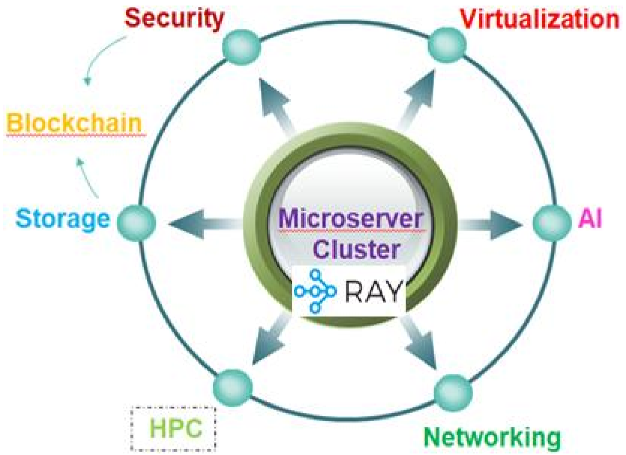
4.3 Re-design and re-implementation of Ray

Pros

- A high-performance distributed execution framework targeted at large-scale ML and RL applications;
- A unified framework for scalable Distributed Computing;
- A fast-growing ecosystem;
- Python-first;
- …

Cons

- Mainly targets X86, and is not well-tested on ARM and RISC-V;
- Focus on Cloud-side, while its potential for Edge-side need to be further released;
- Currently can only be accelerated by GPU, and is not all the XPU-aware;
- A lightweight solution when compared with Spark-based projects, but is also getting heavier;
- …



Source: "Ray - A Swiss Army Knife for Distributed Computing & AI", Feng Li, COSCon 2022.

Ray as a universal infrastructure for distributed computing, especially in HCI (Hyper-Converged Infrastructure)…

II. Practice & Exploration

Our Rayll Series

Rayll.Rust (has some work done)
Rayll.Java (upcoming)
Rayll.Acton (focus on replace Haskell with Zig in the compiler of Acton firstly)
Rayll.Net (has some preliminary work done)
Rayll.Graal (long-term)
Rayll.Zig (long-term)
Rayll4HCI (in the design and early experimental stage)
…

III. Future Work

1) Next generation system language

Ideas:

A desired next generation system language for HW-SW co-development

- Comparable performance to C/C++/Rust/Go.
- Guarantee memory-safety and thread-safety as Rust.
- Low learning curve while high productivity, especially when compared to C++/Rust, and close to the level of Python/Java is mostly preferred.
- Support Multi-paradigm programming, especially for Functional and Metaprogramming, as well as Concurrent.
- A good fit for AI-oriented programming.
- Built-in support for Heterogeneous Parallel Computing.
- Natively support for Distributed Computing.
- Built-in support for Concurrency that closer to Go.
- Self-hosted compilers are mostly preferred.
- Self-contained cross toolchain, build system, unit test and more like Zig
- Good Interoperability for most of the popular programming languages.
- The ability of Bare-metal programming.
- Can be used for Linux Kernel development, and more.
- Easier to be implemented or extended as hierarchical DSLs(Dialects).
- With high-level abstraction ability for HW/SW Co-design, Co-development, Co-simulation, and Co-debugging etc.
- Come with a certain foundation of Ecosystem.

Currently, our technology roadmaps are mainly inspired by Python, Red, D, Zig, Rust, Nim and Mojo.

Source: "First exploration of D for HW-SW co-designed system", Feng Li, 1st OSEDA Workshop China 2022.

III. Future Work

Our technology roadmap:

Method1: **Mojo** (if it will be open source in the future)

Method2: **xDSL** (as an open source alternative to Mojo in some extent, and more)

Method3: **D** **Zig**

For more materials, you may refer to our previous talks **"Will D be a better system programming language"** at OpenInfra Days China 2022 and **"First exploration of D for HW-SW co-designed system"** at 1st OSEDA Workshop China 2022 and the upcoming follow-ups.

…

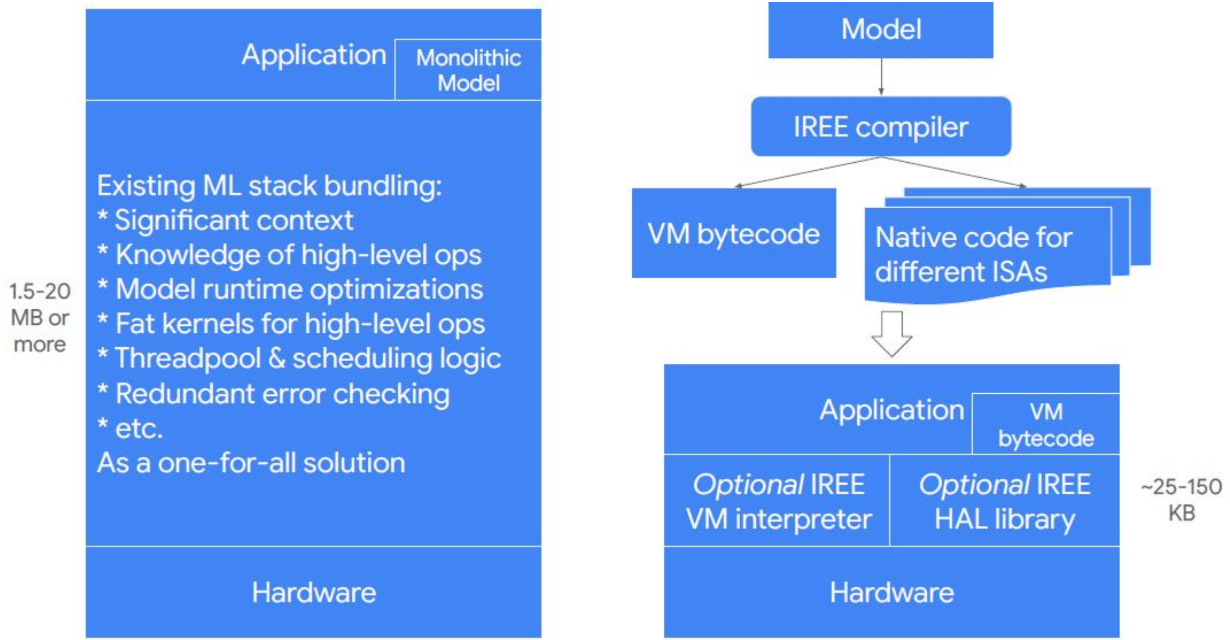
And how about a lightweight re-implementation of LLVM?

III. Future Work
2) New VMs for emerging workloads

IREE Runtime:

IREE does not have a traditional "fat" runtime that bundles everything.

IREE provides an almost zero-cost virtual machine for interpreting host scheduling ops compiled from ML models. It just performs lightweight math for workload size calculation and performs task scheduling.



Source: "IREE: standard-/compilation-based ML stack via Vulkan/SPIR-V", Lei Zhang, Khronos ML Webinar 2022.

Rethinking the architecture & design of new virtual machines for emerging workloads…

IV. Wrap-up

- **Mojo is a great innovation which indicates AI-driven programming language in the future and how they will interact with modern toolchains such as LLVM.**
- **The deep superposition of the ecosystem of Python and LLVM will far beyond our imagination, especially for the field of AI-assisted hardware design and verification.**
- **There will be a golden age for Python-based DSLs in FOSS EDA.**
- **Python & its DSLs are changing the way we think about hardware development!**

鲜卑拓跋枫

扫一扫上面的二维码图案，加我微信