面向RISC-V指令集的操作系统

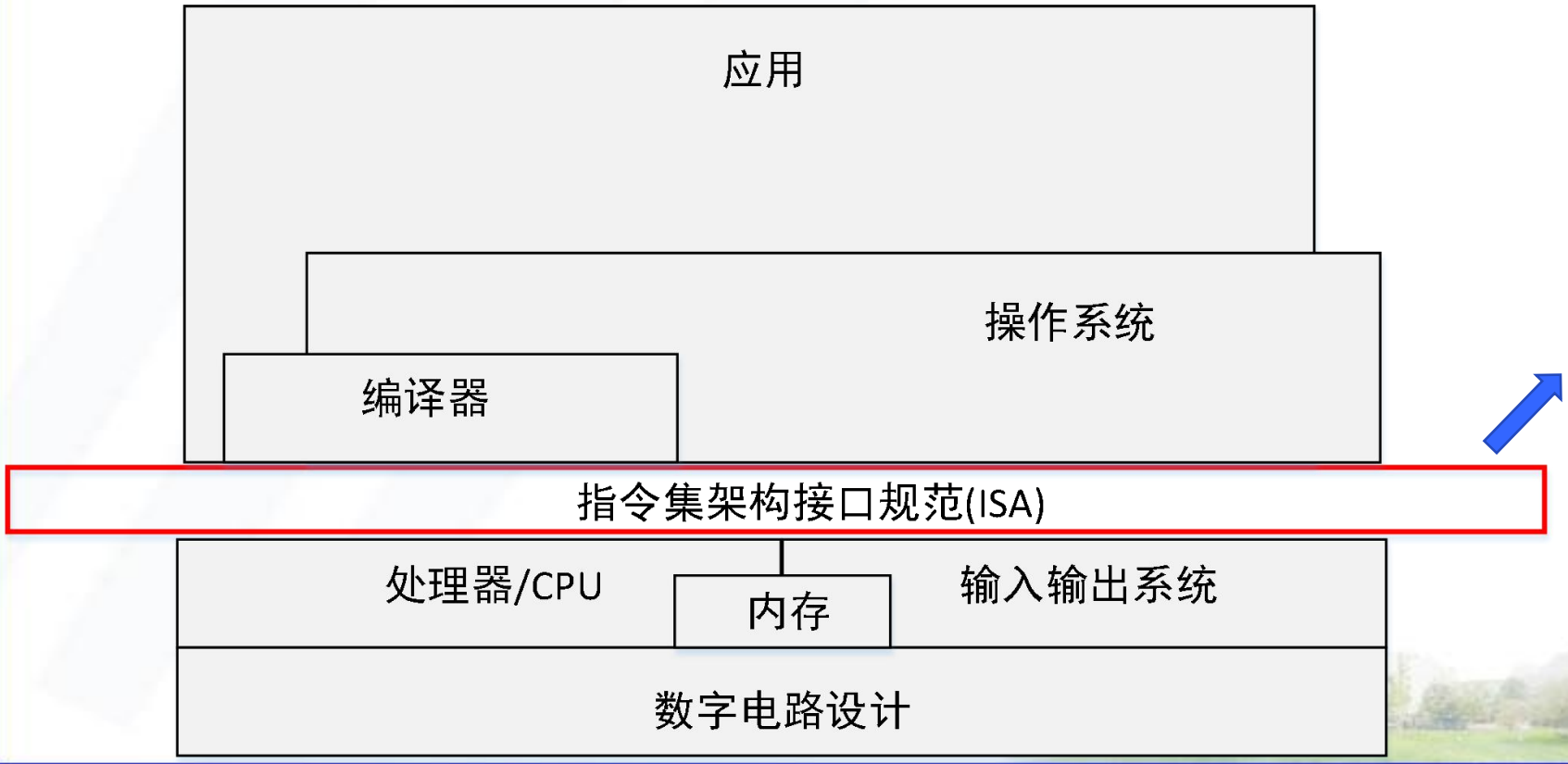
中国科学院软件研究所
汇报人：武延军
2019年12月

提纲

- 一、为什么要做RISC-V操作系统
- 二、RVOS的当前工作进展
- 三、未来工作展望

RISC-V指令集接口规范的机遇

- 指令集作为处理器与系统软件的接口规范，实现了软硬件设计和开发的解耦
  - 前期并行开发，后期融合优化



The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA
Document Version 20190608-Base-Ratified

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
June 8, 2019

RISC-V优势

- **技术先进：指令集架构精简、模块化、易扩展、易读、高效低功耗**
- **指令集开源且开放**
  - 允许任何人或组织自由使用、设计、扩展、实现，开发芯片，以及开发软件，而不必支付指令集专利费
  - 基于**RISC-V**指令集的任何实现，可以自由选择开源、私有、或商业
  - 在当前贸易战暴露出当前全球芯片生态问题背景下，显得尤为珍贵
- **有望形成良性自驱动迭代：形成了成熟的社区组织模式，参与范围广，社区贡献积极专业，生态发展迅速。**

HUAWEI Alibaba Group 阿里巴巴集团 ISCAS SANECHIPS 中兴微电子
SiFive MICROCHIP Google 君正 Ingenic ICT 中科院计算所 IBM

操作系统适配面临挑战

- RISC-V是通用指令集架构有广泛的应用前景
  - 不同于GPU、NPU、MLU等专用加速芯片
  - 因此技术上，RISC-V可支持桌面、服务器、边缘计算、物联网等场景
  - 可以在RISC-V基础上构建桌面操作系统、服务器操作系统、终端操作系统、以及面向边缘计算、AIoT等场景的专用操作系统

- 基础软件支持不足：当前RISC-V的配套基础软件、运行时环境、操作系统支持不足
- 操作系统研发滞后：国内外焦点都在RISC-V硬件，对操作系统的要求只是“能用、够用”，对“好用”的设计与开发关注不足
- 碎片化：由于缺乏统一的操作系统定制规范，正面临碎片化，生态主导作用暂时缺位

为RISC-V打造原生操作系统、编译器，是生态发展的迫切需求。

操作系统应该有新的架构和新的思路

- RISC-V指令集和硬件定制化: 灵活可定制性，伴随硬件碎片化预期
- 计算场景多样化: 操作系统数量爆炸、体积臃肿、系统分裂
- AIoT万物互联场景融合: 边缘、终端、网络、云服务器…

- 这为整体上从头思考操作系统的架构形态带来了新的发展机遇

操作系统需要在架构设计、系统形态、构建模式等方面提出新思路

从头思考操作系统的开发部署模式

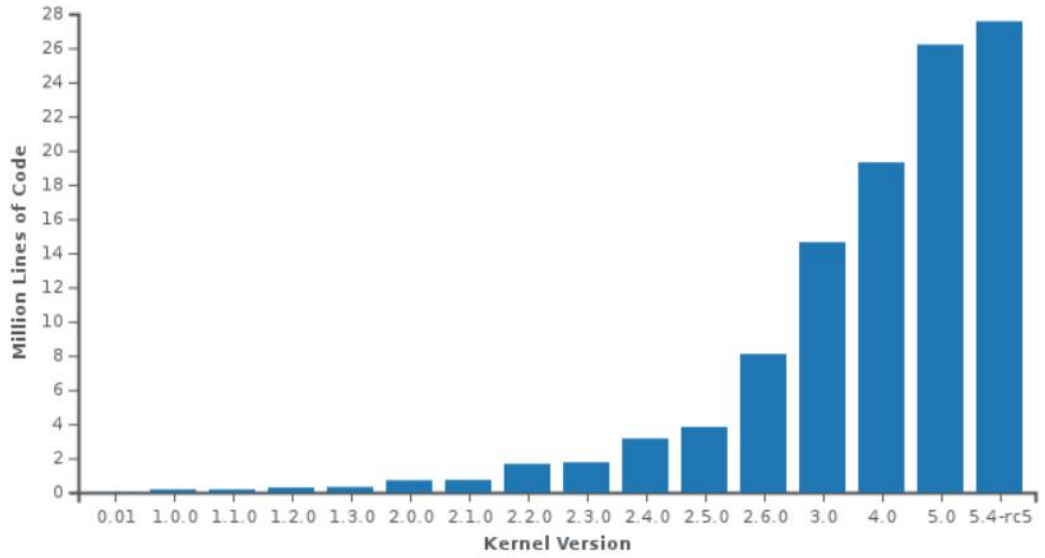
- **OS发展缺乏架构上的整合设计**
  - 操作系统需要总体上做整合设计
  - 避免Case By Case的数量扩张

横向：操作系统数量扩张 1276

纵向：操作系统源码体积增量扩展

- **OS研发构建仍然重量而困难**
  - 个性定制、场景多样的OS
  - 研发敏捷、轻量、灵活

- **OS的功能迭代仍然非常低效**



Linux Kernel

操作系统的架构和研发模式甚至不能满足硬件敏捷定制性和AIoT场景多样性需求

提纲

- 一、为什么要做RISC-V操作系统
- 二、RVOS的当前工作进展
- 三、未来工作展望

1、基于开源软件供应链的操作系统定制方法

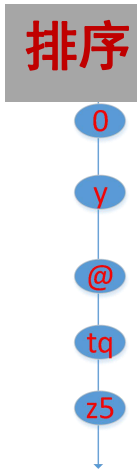
一款商业成功的操作系统为什么这么难？

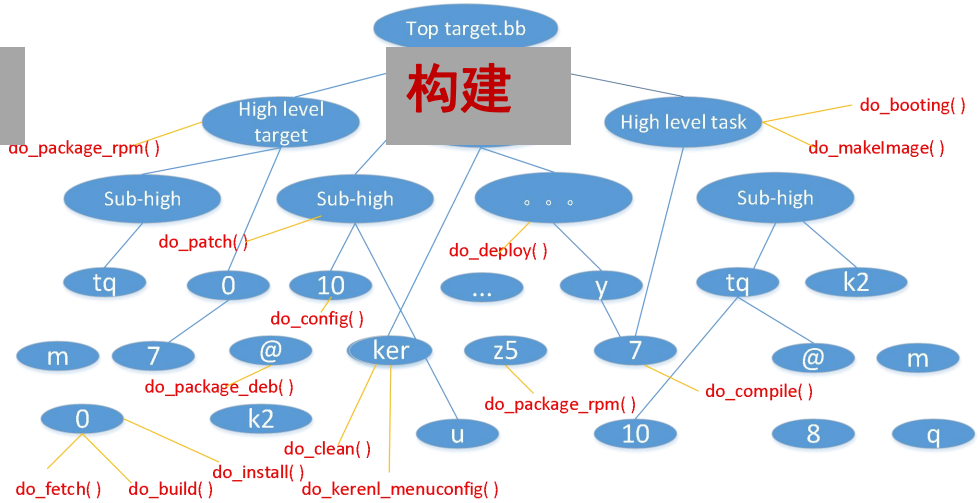
什么是开源软件供应链？

操作系统名称	涉及软件包总数
Ubuntu 18.04	29207
Debian Unstable	32453
FreeBSD Ports	31473
Kali Linux Rolling	27330
Ubuntu 16.04	26910

软件供应链：在开发和运行中用到的软件包集合

可靠开源软件供应链是大规模商用操作系统的基础

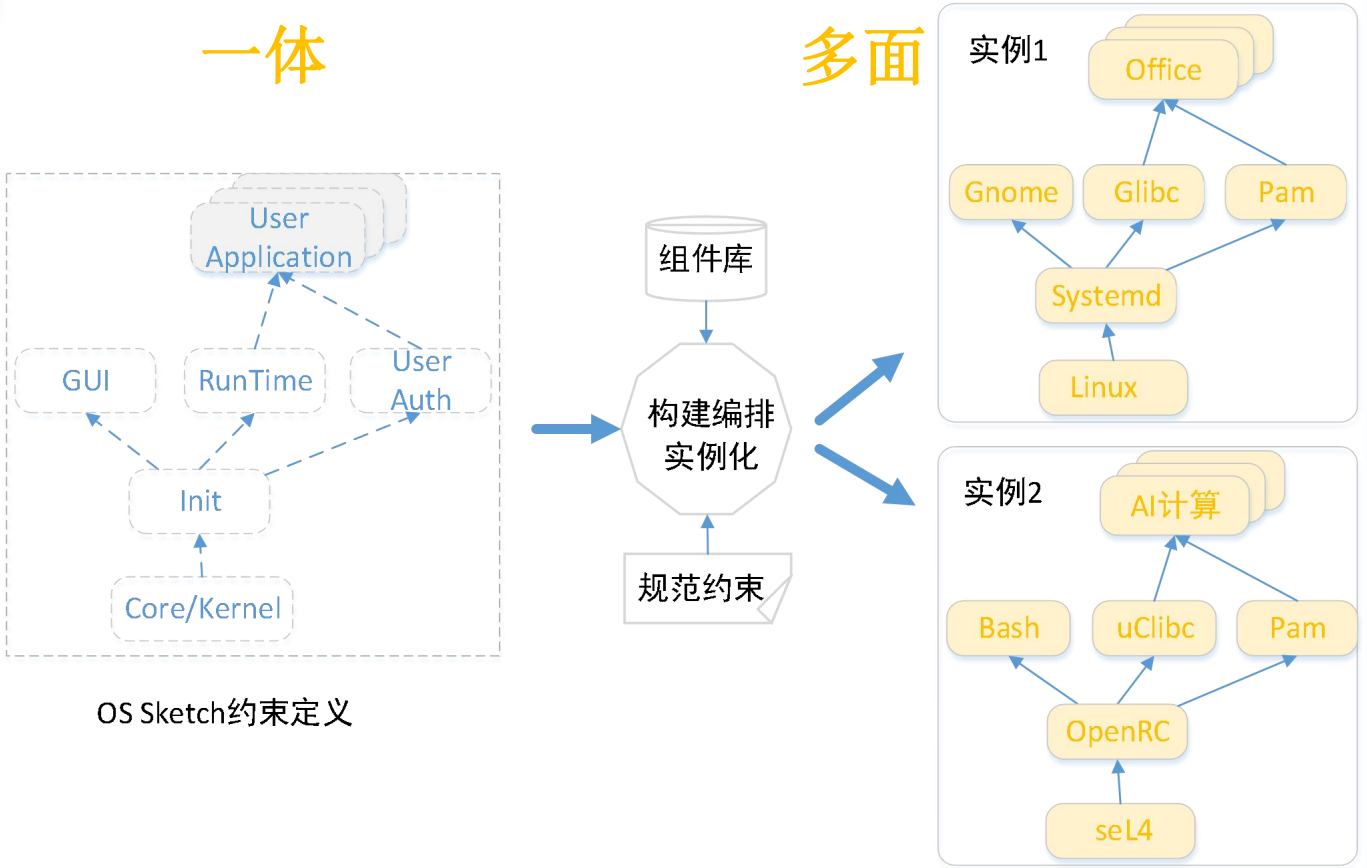




开源软件供应链技术是RedHat价值340亿美元和GitHub价值70亿美元的基础

一体多面OS开发与部署框架设计

- **一体多面：根据智能终端、车机、大屏、工作站、边缘节点、集群等不同特点，呈现最佳适配面的统一操作系统架构。**
- **关键技术特点：灵活**
  - 框架(OS Sketch)可定义
  - 组件化拼装
  - 组件平行可替换



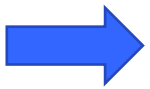
可运行在概念架构上的OS：在抽象架构上设计、开发、运行，并实例化

概念架构示意图：**OS Sketch定义OS结构，选择平行组件填充Sketch,形成OS不同实例,适应多样计算场景**

操作系统框架定义到实例化的技术方案

- 构建定义语言(sDSL)：精确定义操作系统的框架、指挥编排构建
  - 将操作系统实体概念定义为编程语言的内置符号
    - app、driver 、init、lib、daemon、kernel…
    - acpi 、audio 、framebuffer 、gpio 、gpu、input、nic、rtc、 block…
    - …
  - 定义OS的抽象概念架构（OS Sketch）
  - 将操作系统的设计/架构 跟 操作系统的素材（软件包）相分离
- 框架胶水代码：描述对接接口，实现Sketch跟软件供应链素材链接
- 动态链接器(LD)：将object按照不同Kernel的程序镜像布局进行链接

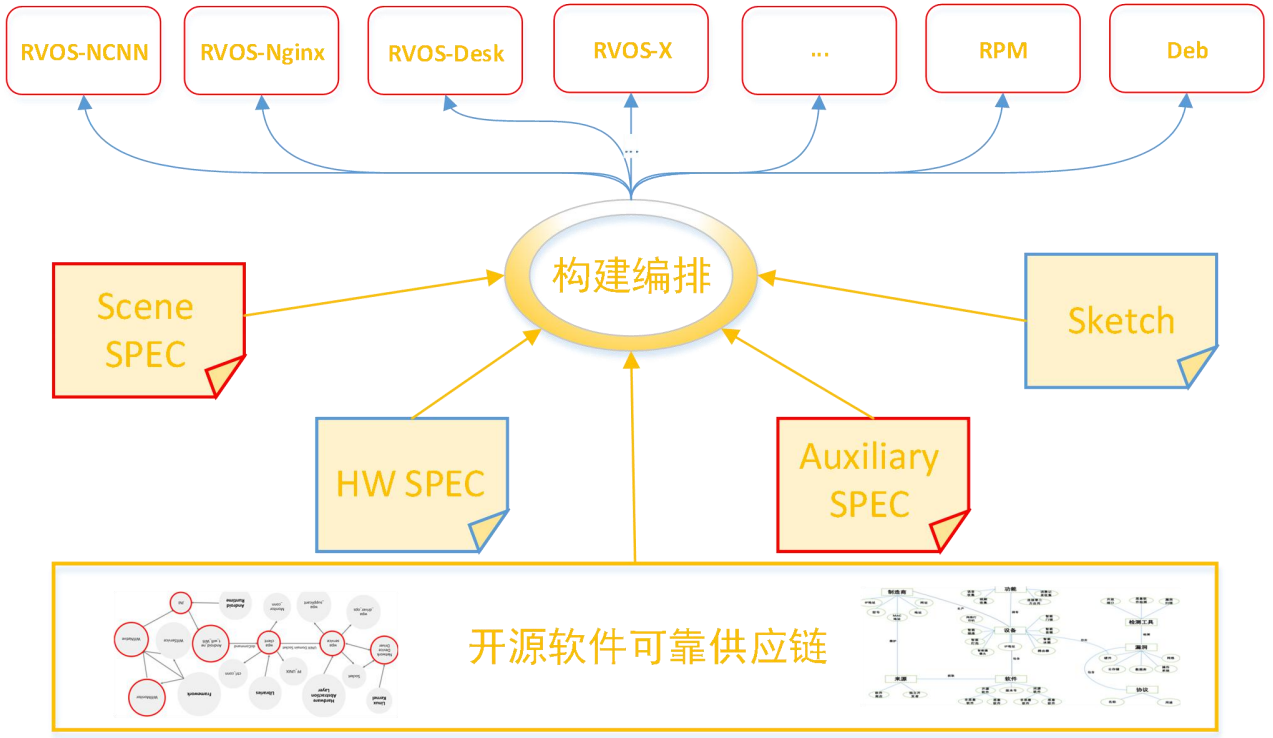
✓ 概念架构以设计文档形式存在
✓ 代码编程实现跟概念设计脱钩
✓ 只有指导意义，没有强制约束



✓ 概念架构的设计过程属于操作系统编程实现的一部分
✓ 设计与实现无缝对接
✓ 设计开始就纳入版本管理

操作系统构建编排环境

- **Scene Specification：操作系统构建中，引入场景描述规范约束**
- **HW Spec：OS的硬件约束规范**
- **OS Sketch：操作系统的框架定义**
- **开源软件可靠供应链：维护组件管理元数据，提供可靠的源码和软件包供应。**



开源软件的供应保障

- 提供对功能模块的组件化管理，分析维护组件关系多维知识图谱，提供可靠的软件包材料服务。
- Safety：独立第三方、供应安全
- Availability：功能完善、运行稳定、性能良好、持续维护
- Alternativity：存在可替代软件、不同场景需求的可选性
- Security：信息安全、漏洞响应

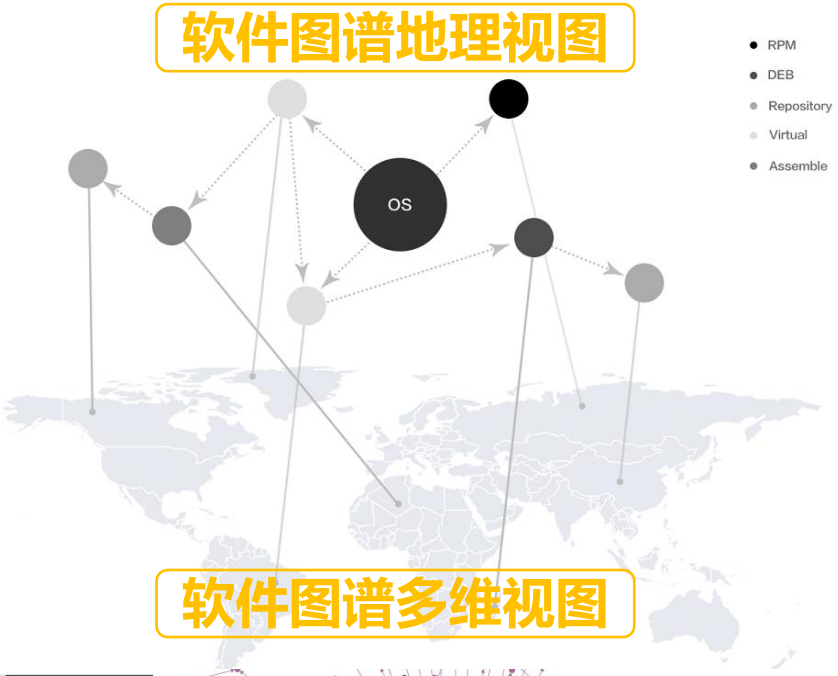
特定产品的
操作系统技术需求

开源软件供应链支撑

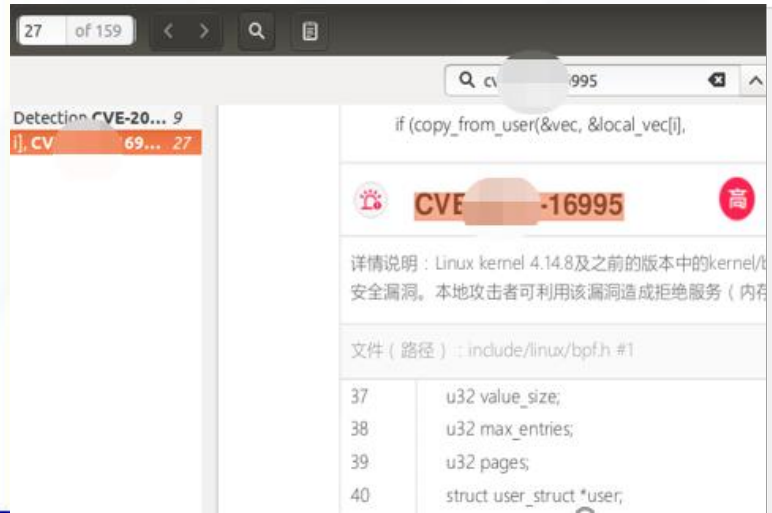
开源软件可靠供应链的管理

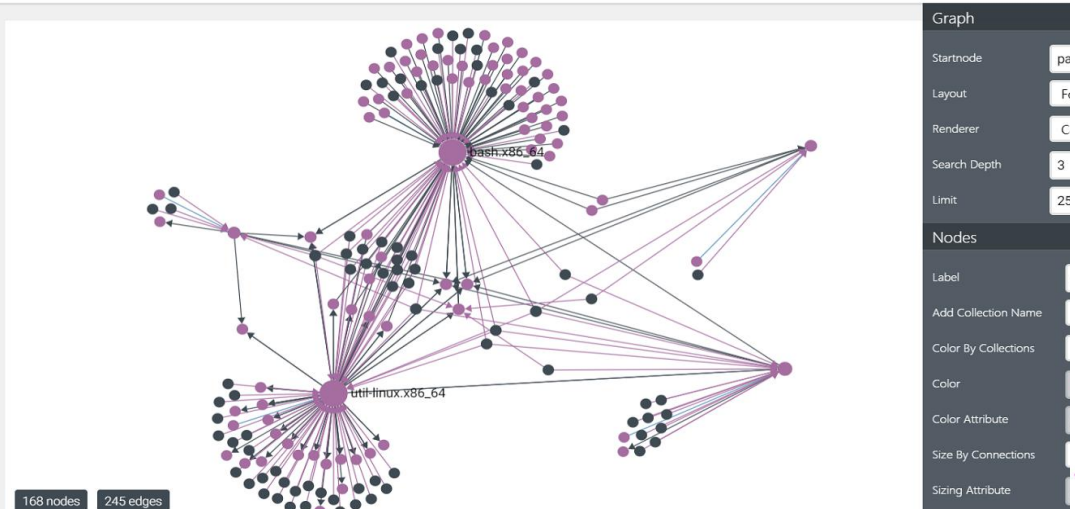
- **建立开源软件知识图谱：**元数据、分布、依赖关系、知识产权、维护者等等
- **智能化管理和服务接口：**满足智能化风险评估、智能信息提取、智能RVOS构建等需求
- **智能化聚焦分析：**可选择重要产品的供应链各个环节做价值评估，提升产业水平
- **新技术融入管理：**增加RISC-V等操作系统新技术的注入效率、共享、和透明管理

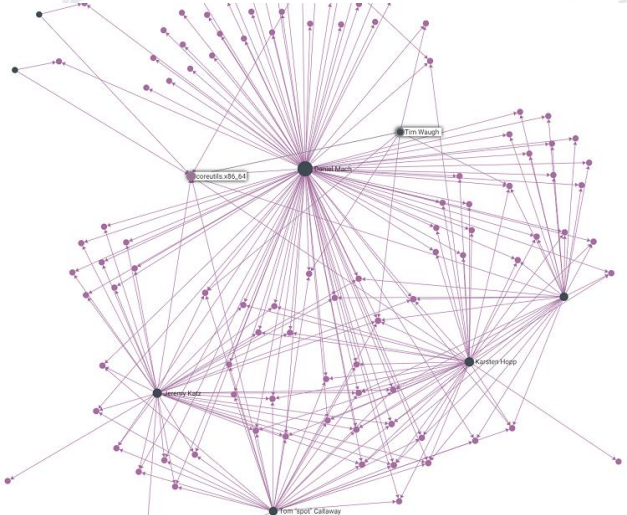
软件图谱地理视图



软件图谱多维视图







操作系统定制样例原型 —— RVOS

- RVOS是面向RISC-V构建的操作系统，致力于成为原生支持RISC-V的的首选操作系统。
- 特征：支持RVOS StemOS、RVOS Yocto嵌入式Linux、RVOS FreeRTOS嵌入式等版本形式，由开源软件可靠供应链提供统一素材保障

- RVOS-StemOS：
  - 探索一体多面新型操作系统架构，实现OS的敏捷快速研发，使得OS定制构建成为一种云服务
  - 面向边缘计算、桌面、服务器、专用垂直定制等
- RVOS-Yocto：嵌入式Linux，满足精细化定制场景
- RVOS-FreeRTOS：面向MCU等嵌入式IoT场景

RVOS——操作系统定制样例原型

- 硬件环境
  - 国内：SERVE
  - 国外：SiFive

表 1 领域专用定制 OS 版本组件数量.

	C/C++ dev	Python RT	Nginx	NCNN[22]	Ubuntu
功能包	9	8	8	10	-
依赖	124	113	111	124	-
总计	**133**	**121**	**119**	**134**	**2030**

- 对StemOS架构和基于软件供应链的定制进行了初步原型验证
  - 领域专用快速定制
    - 敏捷定制：快速构建5个面向专用领域个性化发行版
    - 定制后优势：比其原装软件系统更稳定、镜像可减小90%，跟代表性的Debian、Ubuntu相比，组件数量显著减少
  - 组件平行替换能力
    - 在给定一个OS框架约束定义情况下，分别面向桌面场景和嵌入式场景
    - Linux内核跟seL4微内核
    - uClibc和Glibc

易维护、敏捷可扩展、攻击面窄、安全加固方便

2、LLVM编译器后端支持

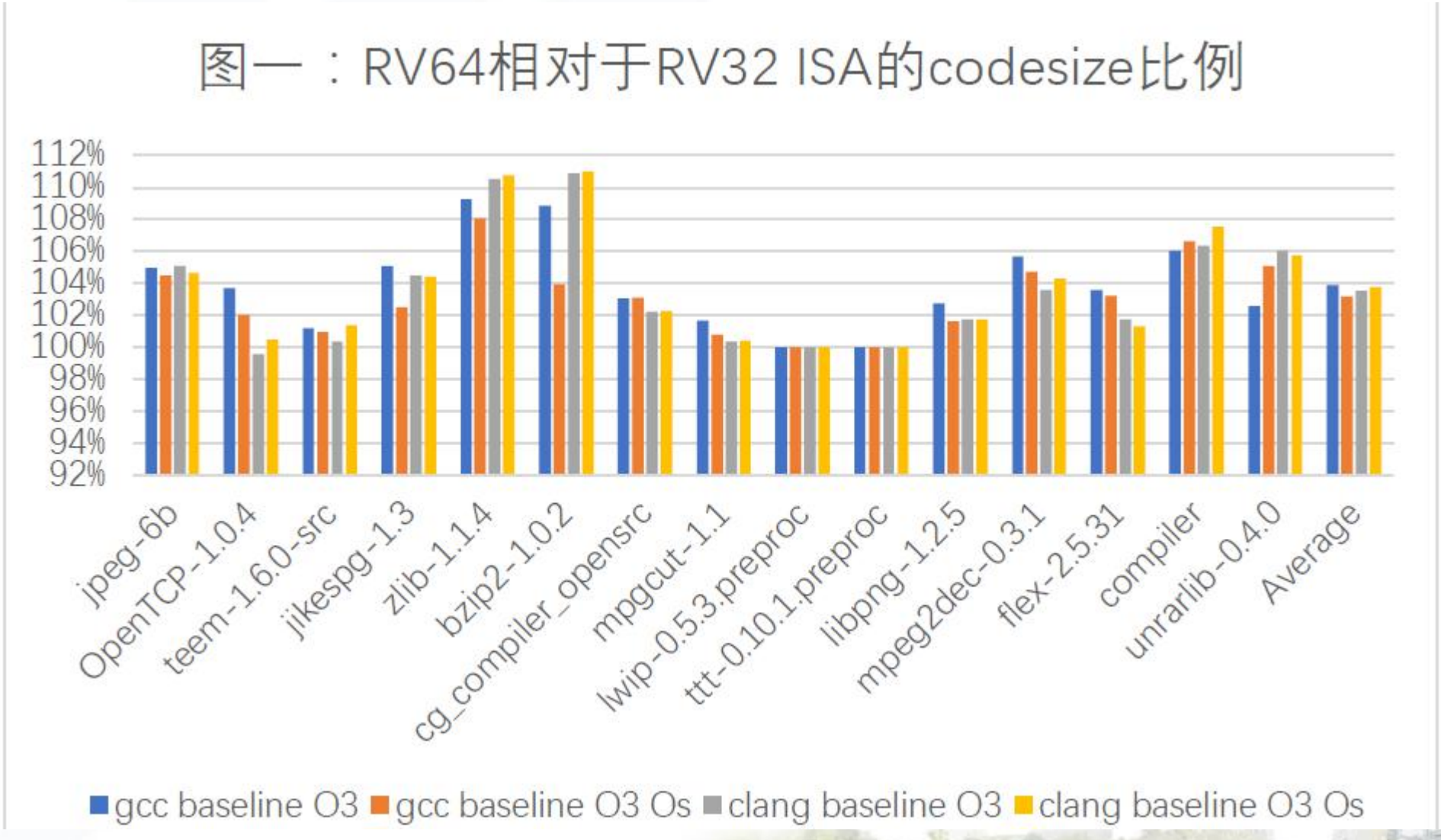
开发进展与愿景/Status Table

- √(unmasked): 338，TODO: 5处，masked 部分多数尚未完成

opcodestr	unmasked(vm=1)	masked(vm=0)
Configuration-Setting Instructions (no vm bit)		
vsetvli	√	
vsetvl	√	
Vector Unit-Stride Instructions		
vlb.v	√	
vlh.v	√	
vlw.v	√	
vlbu.v	√	
vlhu.v	√	
vlwu.v	√	
vle.v	√	
vsb.v	√	
vsh.v	√	
vsw.v	√	
vse.v	√	
Vector Strided Instructions		
vlsb.v	√	
vlsh.v	√	
vlsw.v	√	
vlsbu.v	√	
vlshu.v	√	
vlswu.v	√	
vlse.v	√	
vssb.v	√	
vssh.v	√	
vssw.v	√	
vsse.v	√	
Vector Indexed Instructions		
vlxb.v	√	
vlxh.v	√	
vlxw.v	√	
vlxbu.v	√	
vlxhu.v	√	
vsxb.v	√	
vsxh.v	√	
vsxw.v	√	
vsxe.v	√	
vsuxb.v	√	
vsuxh.v	√	
vsuxw.v	√	
vsuxe.v	√	
Unit-stride Fault-Only-First Loads Instructions		
vlbff.v	√	
vlhff.v	√	
vlwff.v	√	
vlbuff.v	√	
vlhuff.v	√	
vlwuff.v	√	
vleff.v	√	
Vector Load/Store Segment Instructions		
TODO		
Vector AMO Operations		
TODO		
Vector Load/Store Whole Register Instructions (v-0.8 included)		
TODO		
Vector Single-Width Integer Add and Subtract Instructions		
vadd.vv	√	
vadd.vx	√	
vadd.vi	√	
vsub.vv	√	
vsub.vx	√	
vrsub.vx	√	
vrsub.vi	√	
Vector Widening Integer Add and Subtract Instructions		
vwaddu.vv	√	
vwaddu.vx	√	
vwsubu.vv	√	
vwsub.vv	√	
vwsub.vx	√	
vwaddu.wv	√	
vwaddu.wx	√	
vwsubu.wv	√	
vwsubu.wx	√	
vwadd.wv	√	
vwadd.wx	√	
vwsub.wv	√	
vwsub.wx	√	
Vector Integer Add-with-Carry / Subtract-with-Borrow Instructions		
vadc.vvm	reserved	√
vadc.vxm	reserved	√
vadc.vim	reserved	√
vmadc.vv(m)	√	√
vmadc.vx(m)	√	√
vmadc.vi(m)	√	√
vsbc.vvm	reserved	√
vsbc.vxm	reserved	√
vmsbc.vv(m)	√	√
vmsbc.vx(m)	√	√
Vector Bitwise Logical Instructions		
vand.vv	√	
vand.vx	√	
vand.vi	√	
vor.vv	√	
vor.vx	√	
vor.vi	√	
vxor.vv	√	
vxor.vx	√	
vxor.vi	√	
Vector Single-Width Bit Shift Instructions		
vsll.vv	√	
vsll.vx	√	

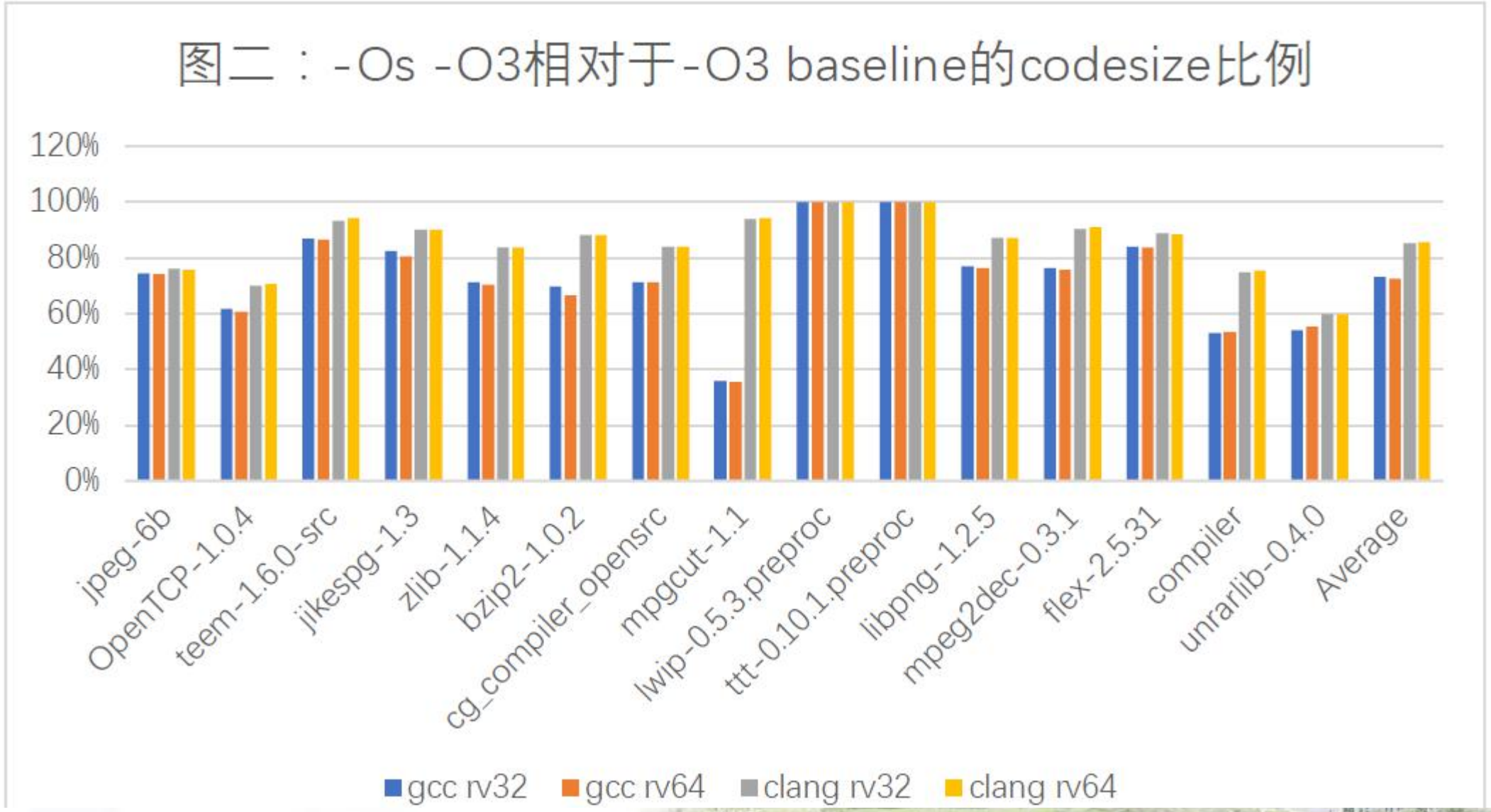
https://github.com/isrc-cas/PLCT-Weekly/blob/master/20191125-rvv-inst-table.adoc

RV64相对于RV32 ISA的codesize比例

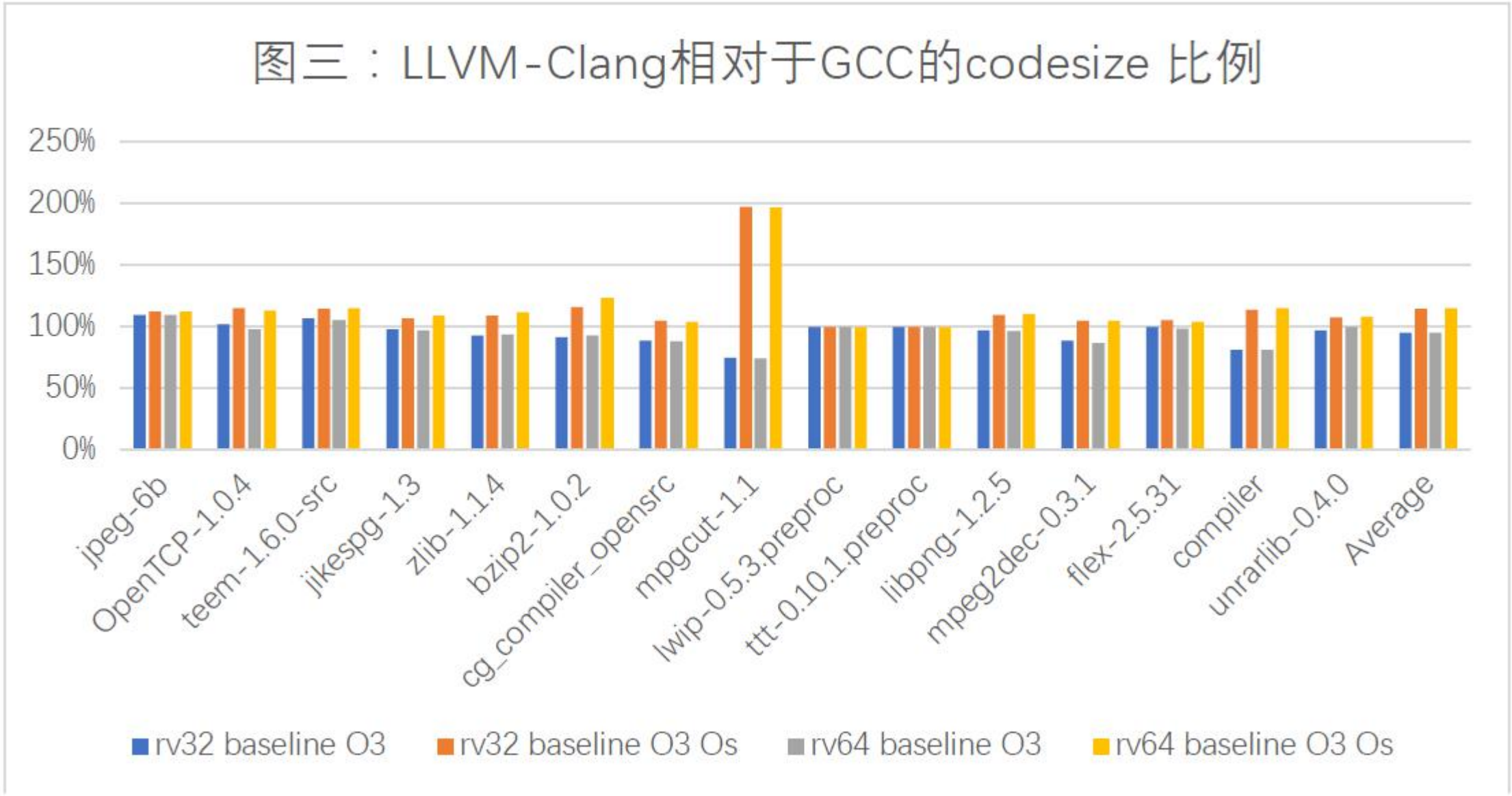


https://github.com/isrc-cas/PLCT-Weekly/blob/master/20191118-RISCV-CodeSize.pdf

关闭和开启Os时codesize 的对比



LLVM-Clang相对于GCC的codesize 比例



有关LLVM的更多内容...

https://github.com/isrc-cas/

3、AIoT类应用场景下的应用和优化方法

面向AIoT的RISC-V深度学习推理框架

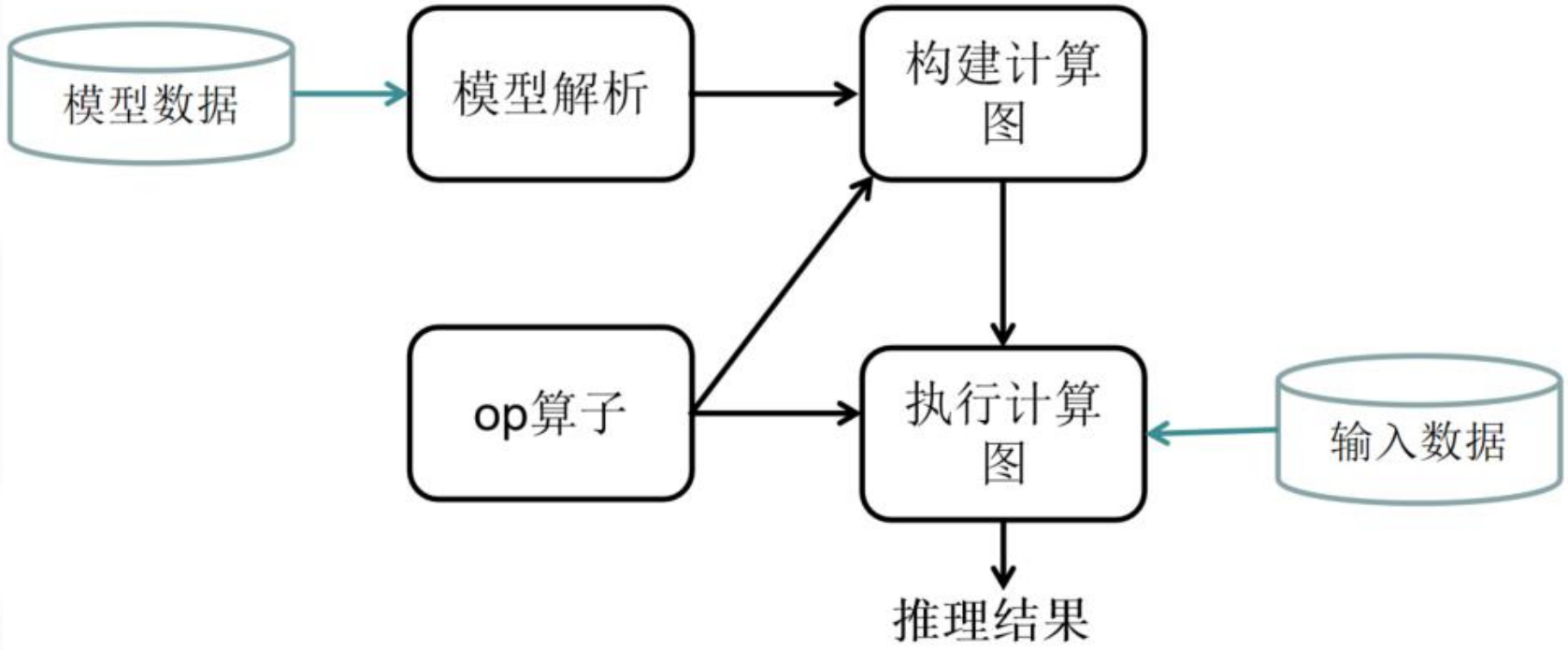
- **RVTensor：RISC-V Tensor**
  - 面向RISC-V + IoT的深度学习推理框架
  - 依赖第三方库少
    - 仅依赖H5模型解析的libhd5.so
  - 内存等资源需求少
  - 支持思沃r版 (SERVE.r) 硬件

全系统平台配置

- Rocket单核/双核@50-100MHz
- UART、GbE、SDIO、USB、HDMI外设
- Linux v4.19 + Debian社区生态
- FPGA定制加速
- 低成本+低功耗板卡

架构与主流程

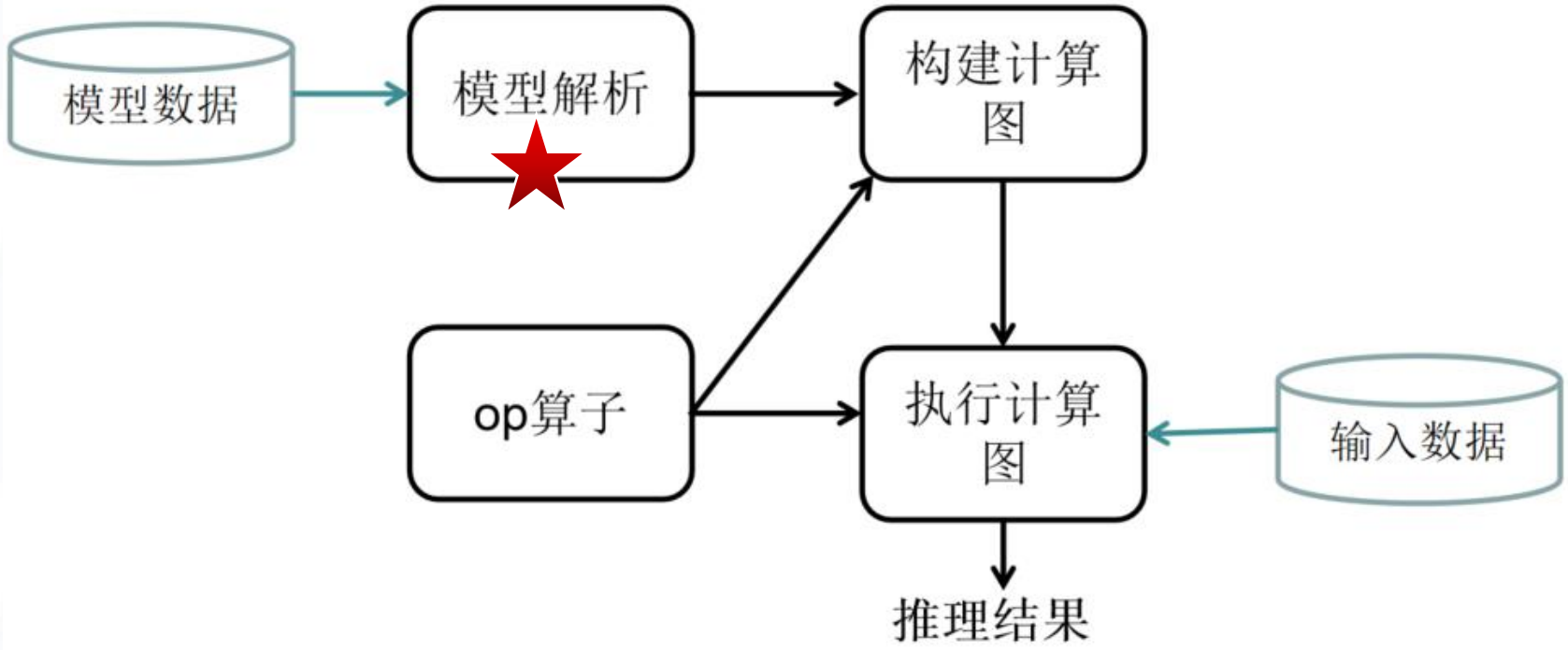
RVTensor架构



- 四个部分组成：模型解析、OP算子、构建计算图、执行计算图

架构与主流程（续）

RVTensor架构

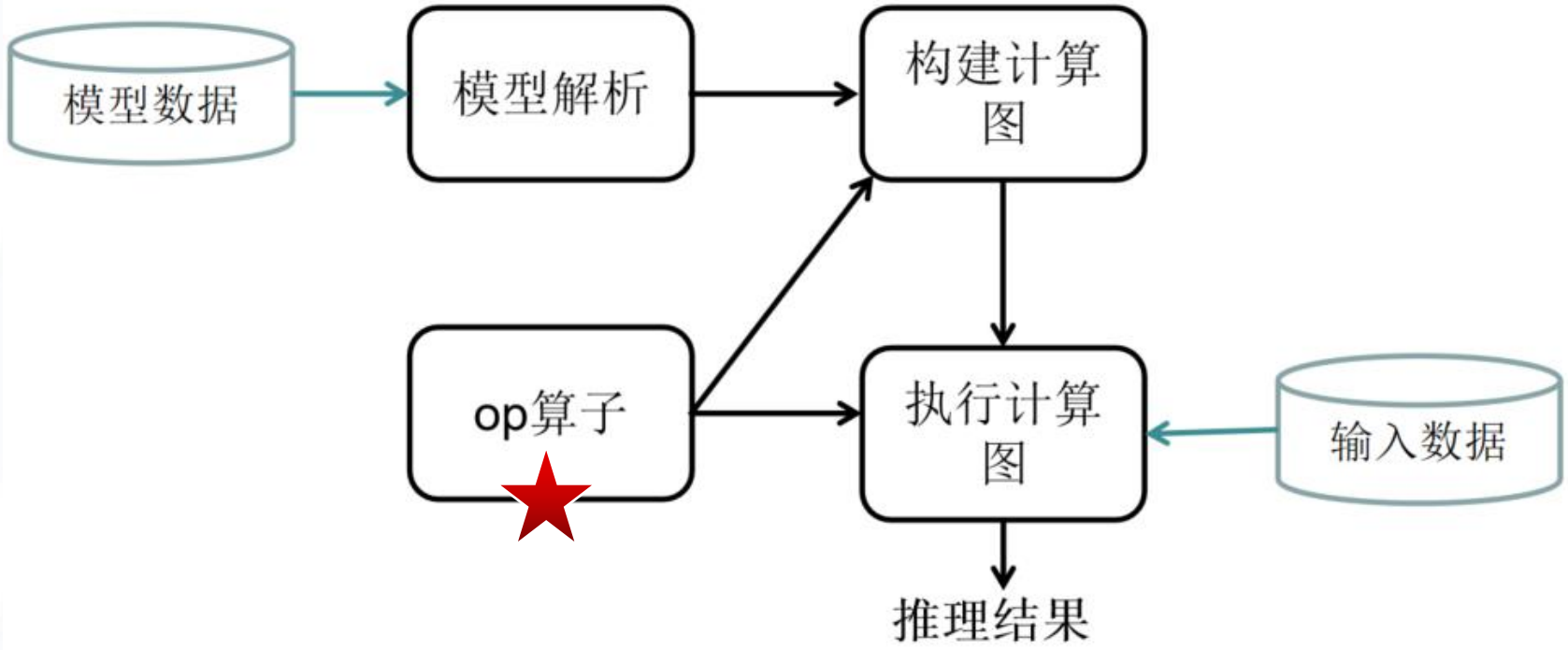


模型解析

- 主要对模型文件如.pb进行解析，读取算子操作、权值数据等信息

架构与主流程（续）

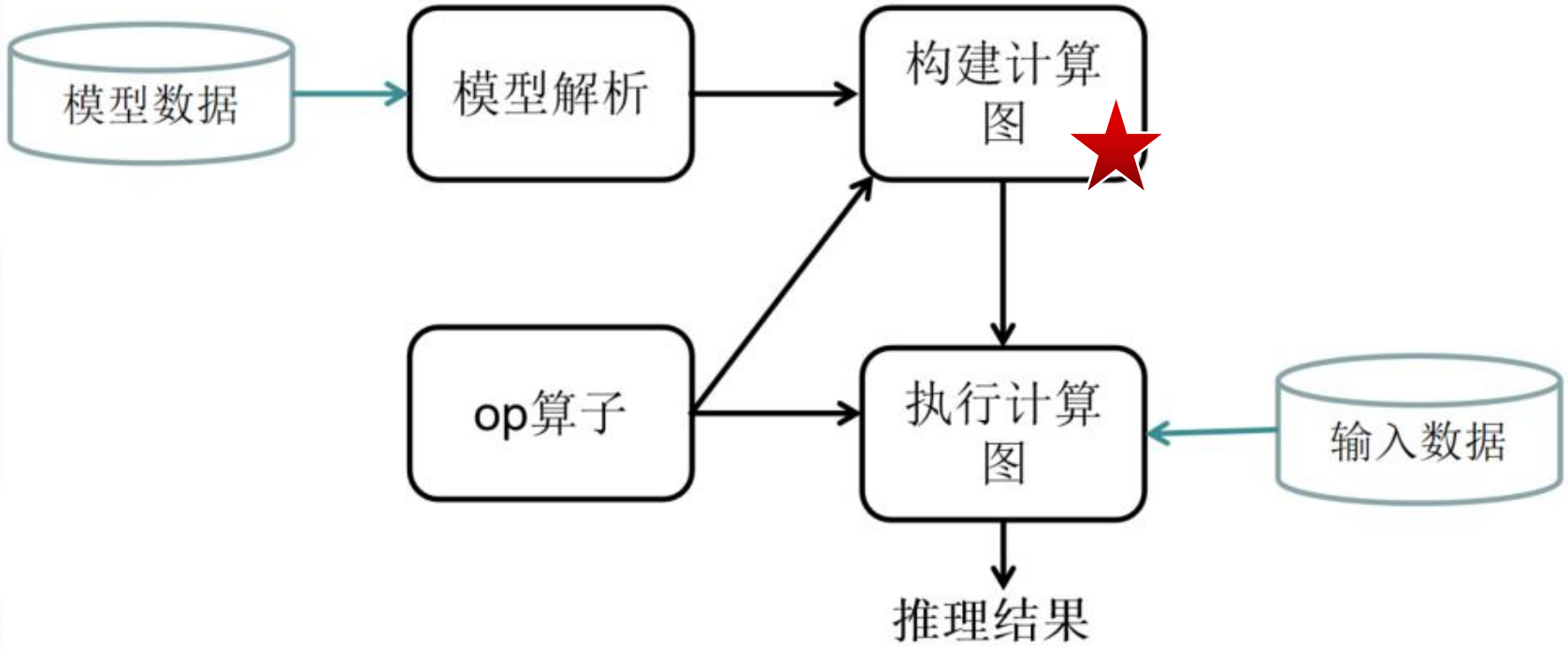
- RVTensor架构



- OP算子
  - 包括conv、add、active、pooling等算子

架构与主流程（续）

RVTensor架构

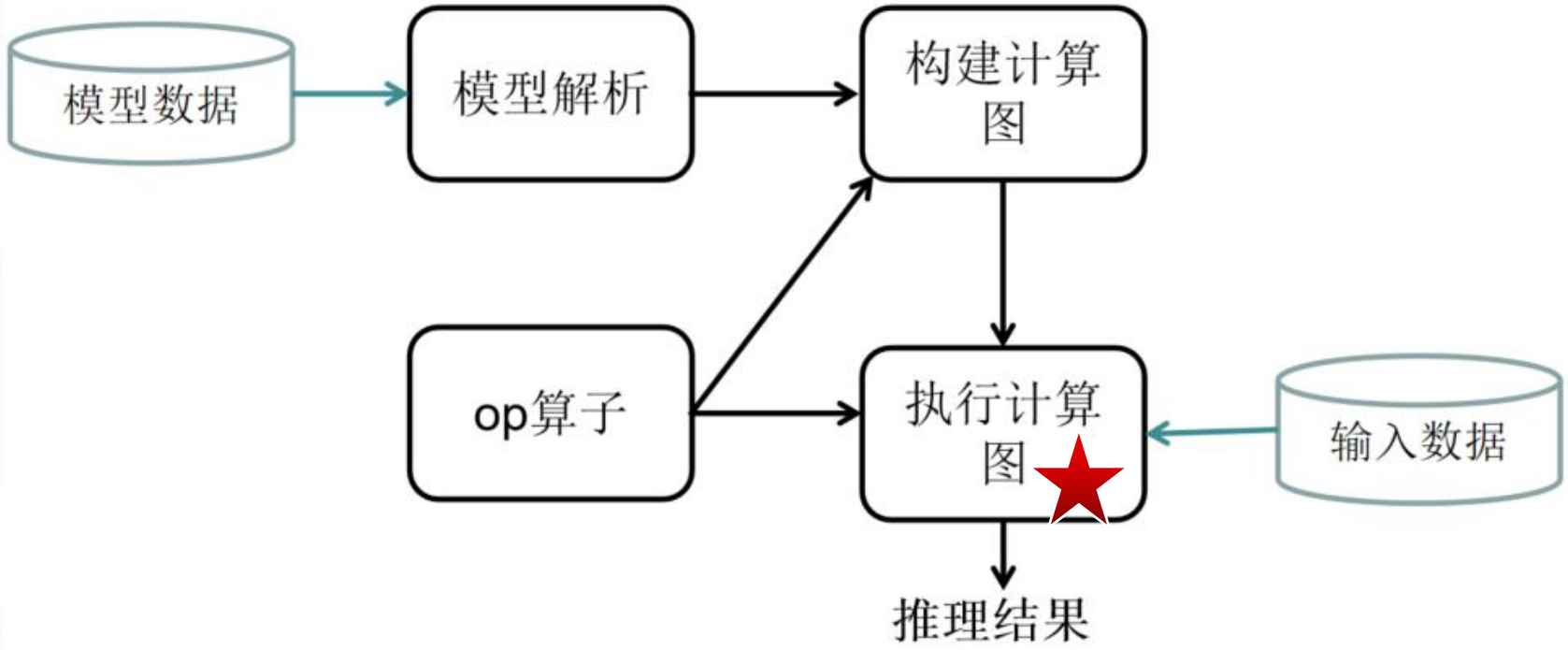


构建计算图

- 基于模型解析和op算子模块构建出计算图

架构与主流程（续）

RVTensor架构

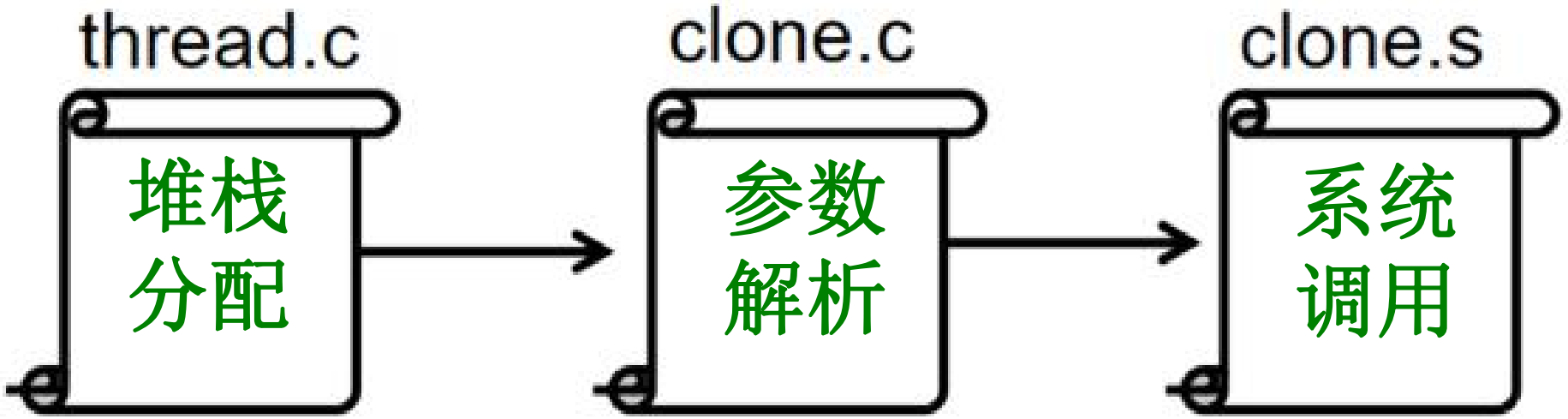


执行计算图

- 该模块基于输入数据和计算图进行计算并得到推理结果

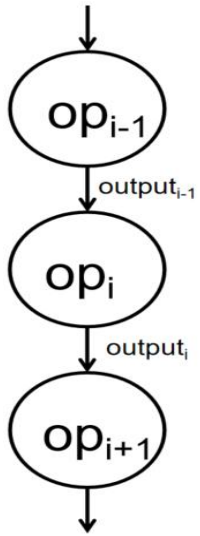
典型优化技术

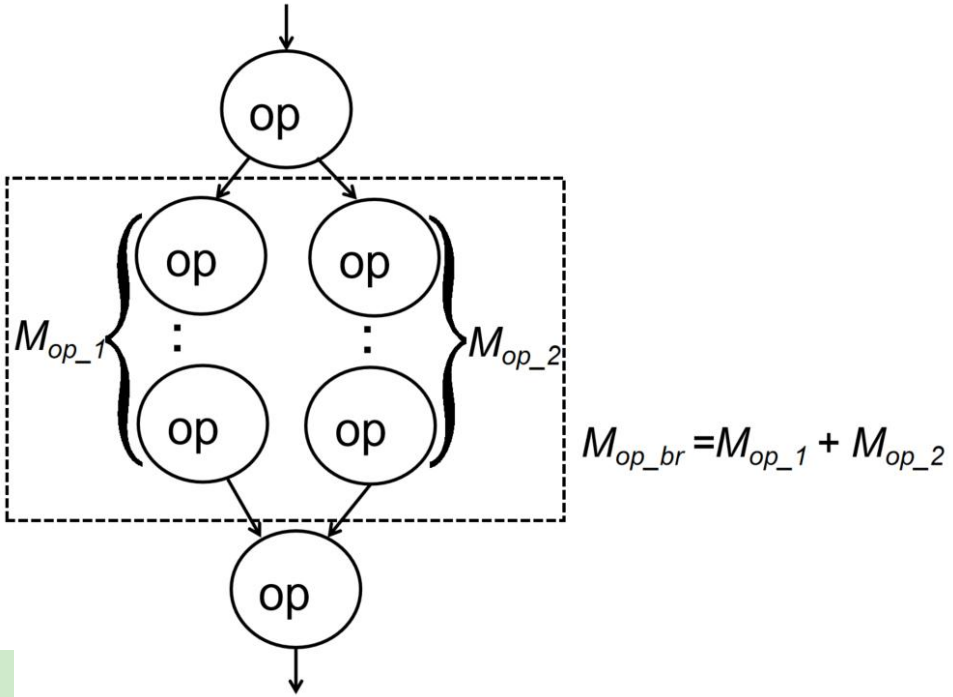
- 针对对第三方库的依赖优化
  - 典型的工作是优化多线程库 **Pthread**
    - 功能很全面
    - 但是用到的**api**很少



典型优化技术（续）

- 内存优化
  - 内存复用：所有op运行时复用同一块内存
    - 最大的op占用内存量
    - 分叉当做原子op

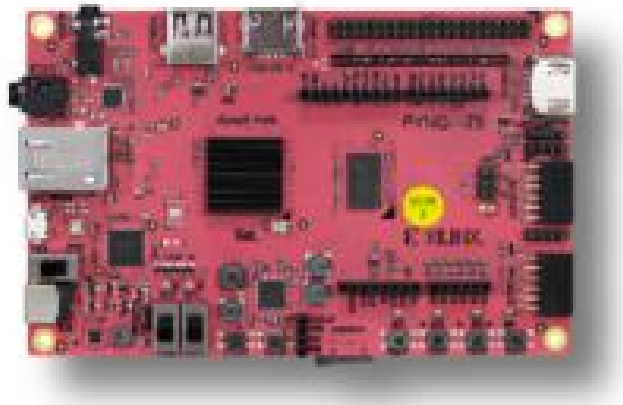


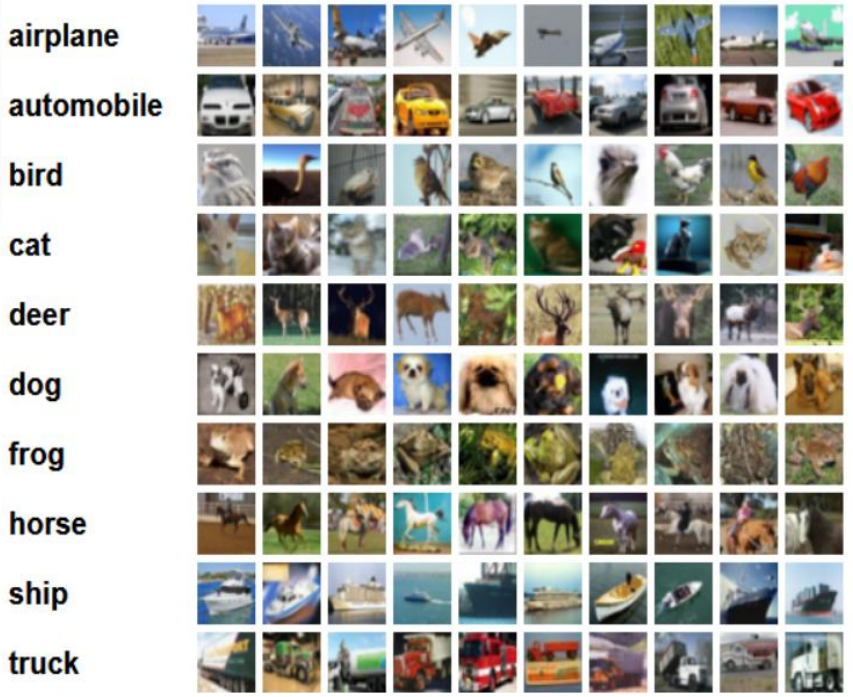


$$M_{op} = \sum_{i=0}^{n} (M_{ii} + M_{ik} + M_{ib}) + M_o$$

优化效果实验评估

- 实验环境
  - 开发板:思沃.r / SERVR.r
  - 测试模型: Resnet20
  - 数据集 : Cifar10





优化效果实验评估

- 准确率
  - RVTensor和Keras的准确率一致

表 2 基于 resnet20 网络模型的准确率统计表

	Top1	Top5
RVTensor	77%	98%
Keras	77%	98%

Keras的准确性评估是基于X86平台完成的

- 性能
  - 处理每张图片的平均时间为13.51秒
- 执行文件大小
  - 193KB

提纲

- 一、为什么要做RISC-V操作系统
- 二、RVOS的当前工作进展
- 三、未来工作展望

未来RISC-V原生操作系统应有的特点展望

- ✓ 原生内核：针对RISC-V指令集架构特点的高度定制化内核
  - ❖ 小核、大核、边缘、智能、桌面、服务器。。。
- ✓ 原生编译器优化：针对RISC-V指令集架构的编译器优化
- ✓ 原生安全架构：面向RISC-V硬件构架，构建安全体系
  - ❖ User Mode
  - ❖ Privileged Mode： Machine Mode、 Supervisor Mode
  - ❖ 。 。 。
- ✓ 原生软件供应链生态：面向RISC-V，形成内容丰富（自给自足）、持续迭代的开源软件可靠供应链
- ✓ 操作系统构建定制进入个性化时代：OS构建能力成为一种弹性云服务，赋能给端用户，满足不同技术层级用户的操作系统快速定制需求

谢谢！

https://github.com/isrc-cas/

欢迎访问和参与