期末作业

- **大作业：40% （团队作业+presentation)**

实验一：系统性能信息采集及可视化平台

- 实验内容：
  - 1.在基于Linux操作系统的硬件平台上，实时采集设备当前的资源及性能数据。
    - 性能数据包括但不限于：CPU数量，CPU核数，CPU占用率，进程数量，物理/虚拟内存大小，物理/虚拟内存占用率，磁盘量，磁盘使用情况及占有率，磁盘IO，网络连接信息，网口流量、IPC（instruction per cycle）、L1/2/3 cache miss等。
    - 技术栈要求：C/C++/Shell；可参考工具：perf stat、vmstat、iostat、memstat等
  - 2.当前硬件设备，能够基于网络获取与之连接的分布式设备的上述数据，并进行可视化平台统一展示。
    - 建议可以直接使用一些可视化框架进行平台搭建，比如grafana、Superset等数据可视化工具，或者结合Highcharts、Echarts等可视化图表库进行平台搭建。
- 评分标准（折算为百分制）
  - 1.（25分）在基于Linux操作系统的硬件平台上，实时采集到设备当前的资源及性能数据。
  - 2.（25分）能够基于网络环境，获取分布式多设备的性能数据。
  - 3.（20分）能够搭建可视化平台，展示上述采集到的性能数据，数据直观且界面美观。
    - 界面设计及实现参考如下页
  - 4.（20分）可视化平台除实时显示采集到的系统级性能数据，可进一步显示不同设备上进程的性能数据（如IPC、CPU利用率、cachemiss等）
  - 5.（10分）书面&口头实验报告

实验一： 系统性能信息采集及可视化平台

- (界面设计及实现参考)

实验二：边缘容器在国产操作系统上的移植与优化

- 实验内容：
  - 1. 在荔枝派4A RISCV开发平台上移植轻量级容器引擎iSula与容器编排系统K8S
  - 2. 分别在Intel X86及RISCV平台上测量并优化isula容器create时间与服务运行时内存占用。
  - 注：课程将提供3套荔枝派4A RISCV开发板。

- 评分标准（折算为百分制）
  - 1. （20分）在荔枝派4A RISCV开发平台上成功运行轻量级容器引擎iSula各项功能
  - 2. （20分）在荔枝派4A 1 RISCV开发平台上成功运行容器编排系统K8S各项功能
  - 3. （20分）测量iSula容器在Intel X86及RISCV平台上的iSula容器create时间与服务运行时内存占用
  - 4. （30分）优化iSula在Intel X86及RISCV平台上的容器Create时间以及服务运行时内存占用
  - 5. （10分）书面&口头实验报告
  - （附加20分）如能将容器create时间或服务运行时内存占用优化10%及以上可酌情附加20分
  - 实验总分不超过100分

实验三-1：容器时序检查点恢复

- 实验内容：
  - 1. 基于开源容器实现时序容器检查点恢复功能

- 评分标准（折算为百分制）
  - 1. （20分）利用CRIU实现容器的检查点-恢复功能
    - 容器选择：isula、containerd、CRI-O等
  - 2. （20分）实现时序的容器检查点保存机制
    - 容器运行情况下周期性checkpoint，恢复时提供检查点选择功能，实现按需checkpoint恢复
  - 3. （20分）实现参考增量式checkpoint，对周期性的容器检查点进行合并或清理
  - 4. （30分）在容器运行时工具现有命令中增加容器时序检查点-恢复命令，或添加命令相关参数
  - 5. （10分）书面&口头实验报告
  - （附加20分）如能将在isulad中实现任务4可酌情附加20分
  - 实验总分不超过100分

实验三-2：isulad容器相关功能实现

- 实验内容：
  - 1. 基于Isulad实现统一的环境依赖检查机制

- 评分标准（折算为百分制）
  - 1. （10分）完成iSulad环境搭建和源码编译
  - 2. （50分）完成iSulad启动环境依赖检查机制
    - 输出markdown设计文档，并提交到社区；
    - 检查点包括：依赖命令、系统配置、内核模块依赖、关键组件依赖（如cgroup是否使能）等等；
  - 3. （20分）功能模块具有可拓展性和可维护性
    - 模块设计应易于新增检查项；
    - 检查错误输出，应易于定位定界；
  - 4. （30分）完成相关功能并提交pr至isulad社区
  - 5. （10分）书面&口头实验报告
  - 实验总分不超过100分

实验三-3：isulad容器相关功能实现

- 实验内容：
  - 1. isula create/run 支持 --gpus 参数，使能容器使用对应的gpu设备

- 评分标准（折算为百分制）
  - 1. （10分）完成容器环境搭建和源码编译
  - 2. （30分）完成isulad支持gpus参数的方案梳理和设计，并输出markdown文档到社区；
  - 3. （30分）完成容器创建及运行支持gpus；
    - gpus参数的客户端isula解析和基本校验；
    - 服务端isulad设置容器使能gpu的环境变量；
    - 服务端isulad设置容器使能gpu的设备信息等；
  - 4. （30分）完成相关功能并提交pr至isulad社区
  - 5. （10分）书面&口头实验报告
  - 实验总分不超过100分

实验三-4：isulad支持线程池

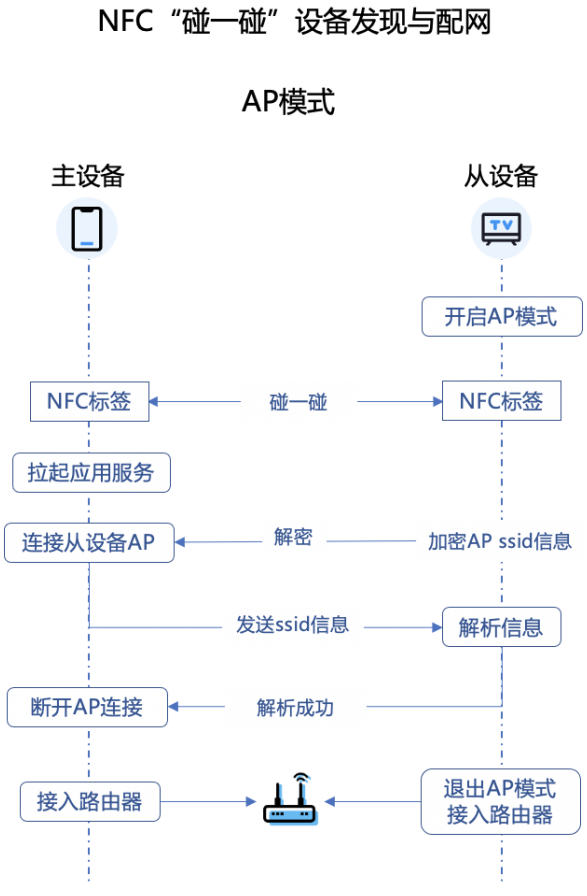
- 实验内容：
  - 1. 为isulad提供统一的线程池，涉及新增线程的场景均由线程池提供

- 评分标准（折算为百分制）
  - 1. （10分）完成容器环境搭建和源码编译
  - 2. （30分）完成梳理当前isulad中线程使用场景，根据当前使用情况完成线程池设计方案，并输出markdown文档提交pr到社区；
  - 3. （30分）完成高稳定和易用的C语言线程池；
  - 4. （30分）完成相关功能并提交pr至isulad社区
  - 5. （10分）书面&口头实验报告
  - 实验总分不超过100分

实验四：基于全志D1平台的设备自动发现与组网

- 实验内容：
  - 1.在全志D1平台上实现基于NFC的设备发现、组网与通信。
  - 2.在全志D1平台上实现基于BLE的设备发现、组网与通信。
- 评分标准（折算为百分制）
  - （20分）在全志D1平台上进行NFC驱动适配，使其支持NFC拓展芯片。
    - NFC适配可基于I2C或SPI接口进行驱动适配和通信。以树莓派PN532为例，具体可参考下述文档：
    - PN532 NFC HAT官方说明文档；树莓派相关驱动文件（包含PN532驱动文件）：驱动依赖文件
  - （25分）基于全志D1平台及NFC芯片，简化手动配网流程，实现设备间基于NFC“碰一碰”的配网与通信。
    - 如：主动设备通过近场NFC通信协议或12C总线协议读取被动设备标签信息，获取网络信息并主动接入，使主被动设备能连接在同一网络环境下，并基于TCP/IP等网络通信协议进行数据传输。参考wpa_supplication
  - （20分）在全志D1平台上进行BLE驱动适配，使其支持BLE功能。
    - 可参考Linux蓝牙协议栈BlueZ：http://www.bluez.org/
    - 如BlueZ中通过HCI层，通过驱动模拟USB设备实现串口通讯，实现蓝牙芯片与主机通讯。
  - （25分）基于全志D1平台，实现基于BLE的设备组网与通信。
    - 可通过蓝牙辅助配网实现，如通过WiFi+BLE的方案，利用BLE的扫描发现和通信能力，将WiFi连接所需的SSID、无线密码等信息传输给WiFi+BLE设备，使设备顺利接入互联网，完成设备间的组网，并基于通信协议进行数据传输。
  - （10分）书面&口头实验报告

实验四：基于全志D1平台的设备自动发现与组网



实验五-1：操作系统优化扩展实验

- 实验内容：
  - 在openEuler上实现Atune的移植
  - 进行Atune的优化扩展
- 评分标准（折算为百分制）
  - 1、（10分）部署运行openEuler系统，并移植适配Atune优化工具。
  - 2、（20分）基于Atune完成系统信息采样扩展
    - 尝试增加相应的采样数据输出，如增加cache miss等性能采集指标，并根据新的采集重新训练模型。
  - 3、（30分）基于Atune实现对pytorch、rabbitmq、mysql等三种应用的调优。
    - 基于当前算法或添加新算法，对应用或benchmark进行训练调优，实现调优参数推荐
    - 对比不同算法和采集数据种类生成模型的调优效率。（比如算法收敛时间，参数空间选择等）
    - openEuler操作系统上进行应用进行性能调优，找到对该应用性能有影响的应用/OS参数
  - 4-1（40分）基于Atune实现系统级的优化。
    - 将系统参数设置作为调优对象，进行系统级优化。如通过proc文件系统、sys文件系统、cgroup等，动态设置系统参数（如调度策略、大小页、中断分发策略、numa设置等）的方式进行系统级性能调优
  - 4-2（40分）面向容器或虚拟化环境实现典型应用的调优
    - 实现对容器应用或虚拟机内应用的调优
  - 5 （10分）书面&口头实验报告

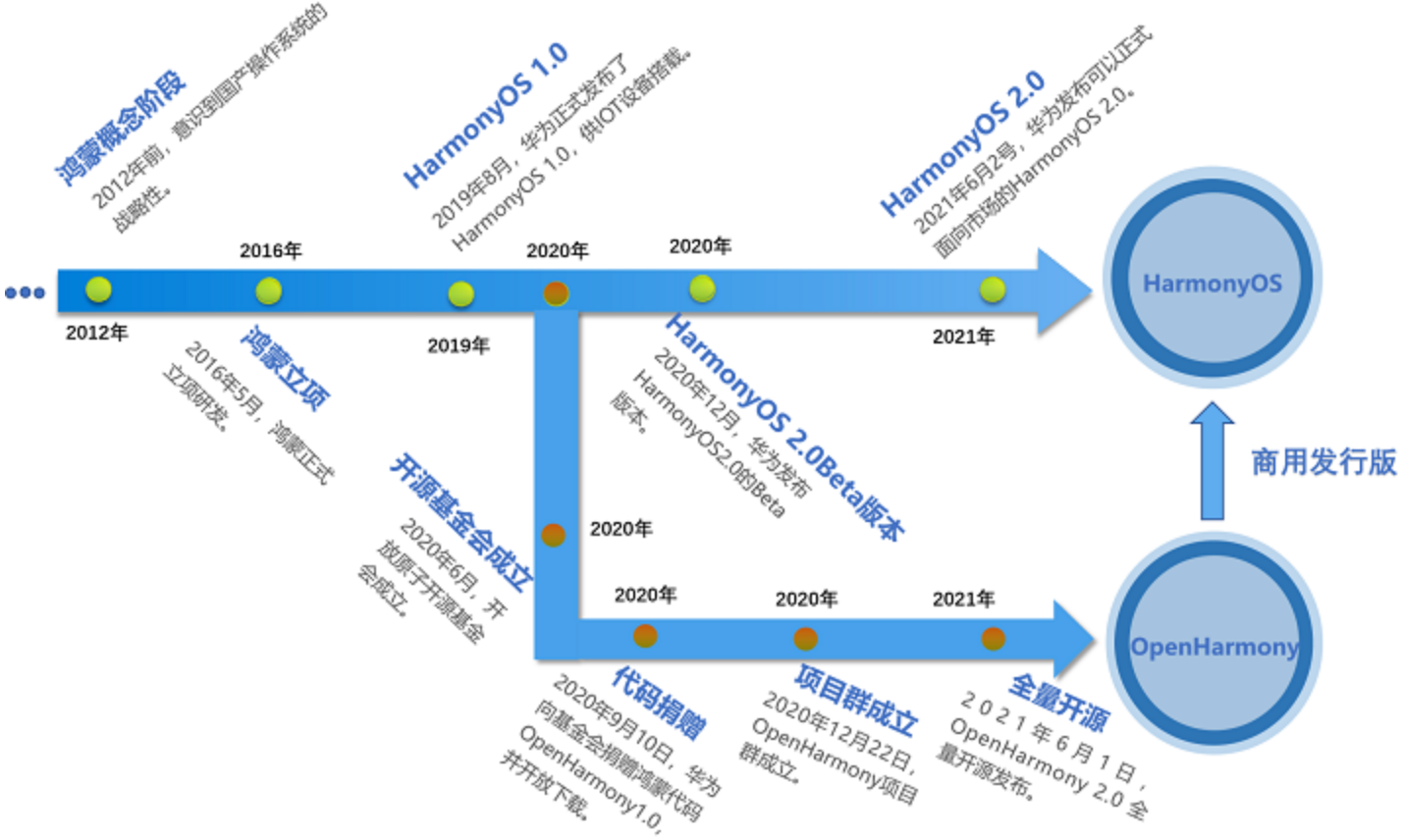
实验五-2：操作系统优化扩展实验

- 实验内容：
  - 在openEuler上实现Atune的移植
  - 进行Atune的优化扩展
- 评分标准（折算为百分制）
  - 1、（10分）部署运行openEuler系统，并移植适配Atune优化工具。
  - 2、（20分）基于Atune完成系统信息采样扩展
    - 尝试增加相应的采样数据输出，如增加cache miss等性能采集指标，并根据新的采集重新训练模型。
  - 3、（30分）基于Atune实现对pytorch、rabbitmq、mysql等应用的调优。
    - 基于当前算法或添加新算法，对应用或benchmark进行训练调优，实现调优参数推荐
    - 对比不同算法和采集数据种类生成模型的调优效率。（比如算法收敛时间，参数空间选择等）
    - openEuler操作系统上进行应用进行性能调优，找到对该应用性能有影响的应用/OS参数
  - 4（40分）根据应用负载特征识别应用类型并提供配置优化参数。
    - 实现软件模型的常见场景感知，如计算、存储、网络等软件负载资源。
    - 实现软件负载的正确分类识别
      - 通过优化现有训练分类模型准确度或通过选择采集性能数据的准确性
      - 或通过启发式策略识别应用，比如监控运行的进程名称等。
    - 实现对应场景下的调优参数推荐
  - 5（10分）书面&口头实验报告

期末作业

- 作业提交：
  - 提交代码Patch或改动的源码文件或在开源社区提交pr、文档（代码说明、复现流程、大作业的技术路线和具体实现，作业过程中遇到问题、bug的解决方法说明、组员分工等）
    - 详细介绍实验过程中遇到的问题，解决的方法需要列入文档并可复现。
    - 组内的分工需在文档中体现
  - 提交课堂报告PPT，并进行课堂报告（抽选，无论是否课堂报告都需提交）
    - 介绍技术路线和具体实现
    - 介绍验证方法
    - 介绍用到的原理和技术，遇到的困难和解决方法，bug和debug，问题，收获.....
    - 组内的分工
- 大作业：40%（团队作业+presentation）
  - 任务提交时间：2024.1.5
  - 大报告时间：2024.1.12

OpenHarmony和HarmonyOS关系



OpenHarmony RISC-V移植进展

工作内容	描述
RISC-V软件栈移植	OpenHarmony 中找出与arm强相关的部分，解绑、增加risc-v的支持
芯片移植	QEMU RISC-V、Dayu800等
应用生态扩充	为OH中的js提供RISC-V的支持

OpenHarmony RISC-V移植进展

OpenHarmony架构强相关软件栈移植

工具链

- 基于LLVM定制的OpenHarmony工具链RISC-V适配

三方库

- 架构强相关三方库适配，如Musl、Libunwind、LibFFI等
- 硬件平台相关的库，如GPU调用库Mesa3D

子系统组件

- 适配与架构相关的上层组件，如方舟多语言运行时（基础适配）等

OpenHarmony RISC-V移植进展

芯片移植

QEMU RISC-V平台

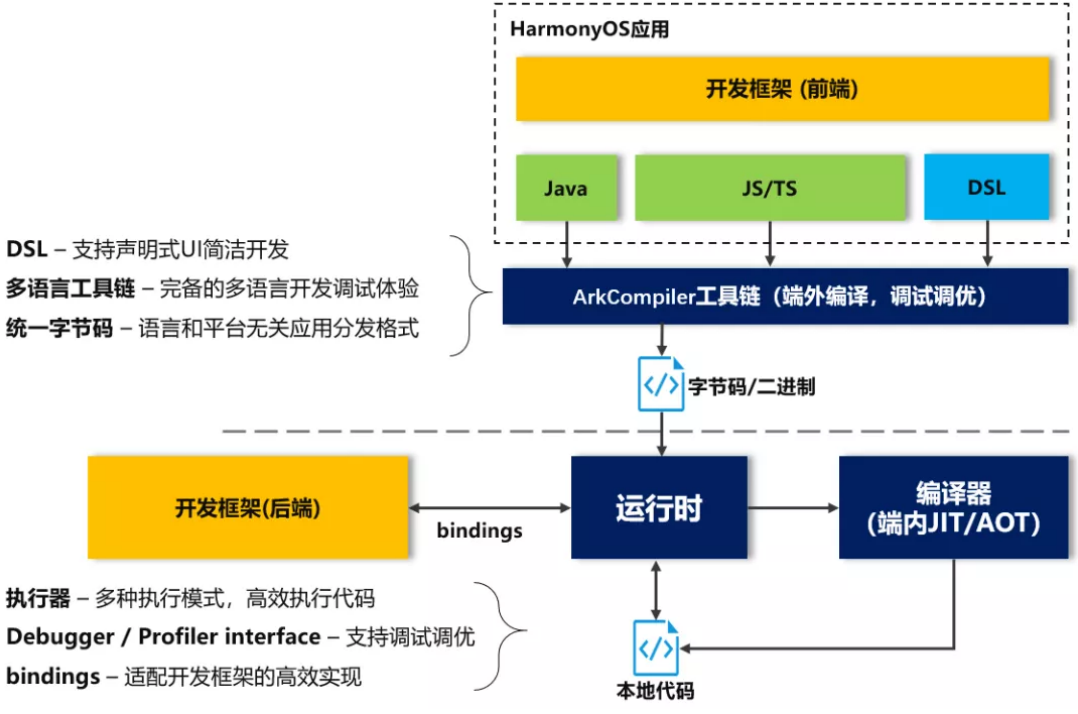
- 完成OH3.2 for RV适配
- 驱动支持：图形、音频、摄像头、网络、键盘、鼠标
- 增加虚拟按键功能

润开鸿DAYU800开发套件(基于平头哥TH1520芯片)

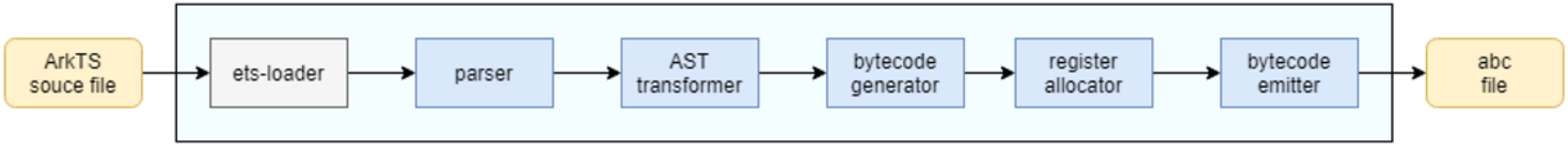
- 完成OH3.2 for RV适配
- 应用方案支持（平板电脑、边缘计算网关、云桌面终端

方舟运行时子系统

方舟编译器(ArkCompiler)是为支持多种编程语言、多种芯片平台的联合编译、运行而设计的统一编译运行时平台。它支持包括动态类型和静态类型语言在内的多种编程语言，如JS、TS、ArkTS；
它支撑OpenHarmony系统成为打通手机、PC、平板、电视、车机和智能穿戴等多种设备的操作系统的编译运行时底座。



编译工具链

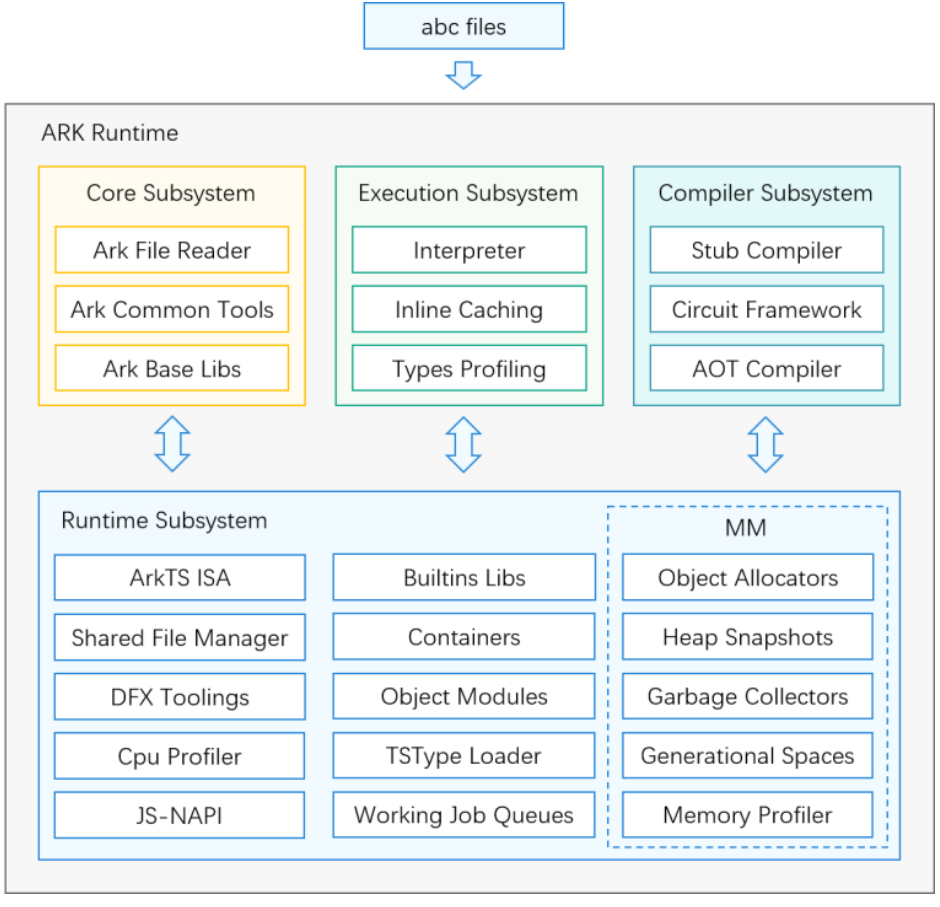


ArkCompiler的编译工具链以ArkTS/TS/JS源码作为输入，将其编译生成为abc(ArkCompiler Bytecode，即方舟字节码)文件。

运行时

ArkCompiler运行时直接运行字节码文件，实现对应语言规范的语义逻辑。

- **Core Subsystem**：主要由与语言无关的基础运行库组成，包括承载字节码的File组件、支持Debugger的Tooling组件、负责适配系统调用的Base库组件等。
- **Execution Subsystem**：包含执行字节码的解释器、快速路径内联缓存、以及抓取运行时信息的Profiler。
- **Compiler Subsystem**：包含Stub编译器、基于IR的编译优化框架和代码生成器。
- **Runtime Subsystem**：包含了ArkTS/TS/JS运行相关的模块。
    - 内存管理：对象分配器与垃圾回收器(并发标记和部分内存压缩的CMS-GC和Partial-Compressing-GC)
    - 分析工具：DFX工具、cpu和heap的profiling工具
    - 并发管理：actor并发模型中的abc文件管理器
    - 标准库：Ecmascript规范定义的标准库、高效的container容器库与对象模型
    - 其他：异步工作队列、TypeScript类型加载、跟C++交互的JSNAPI接口等。



方舟运行时RISCV现状

- C 解释器
  - 利用clang编译器翻译成汇编代码，但是clang编译器无法做到精细的代码分支预测
  - 需要inline的函数没有inline
  - 活跃变量无法寄存器化
  - 目前可以通过C编译器在RISC-V架构上运行字节码
- 汇编解释器
  - 通过Circuit IR手动控制hot代码的路径选择是否inline
  - 利用GHC calling convention支持活跃变量寄存器化
  - 目前汇编解释器只支持ARM64和x86_64

OpenHarmony软件栈RISC-V移植举例

- **汇编和内联汇编**
  - 参考上游社区实现（三方库）
  - 理解ARM、ARM64或者x64汇编实现，实现RISC-V版本（难度高）

```
diff --git a/crt/riscv64/crti.s b/crt/riscv64/crti.s
new file mode 100644
index 00000000..8b54a65d
--- /dev/null
+++ b/crt/riscv64/crti.s
@@ -0,0 +1,11 @@
+.section .init
+.global _init
+.type _init,%function
+_init:
+        ret
+
+.section .fini
+.global _fini
+.type _fini %function
+_fini:
+        ret
```

```
o/adapter/common/include/softbus_adapter_cpu.h
35,6 +35,8 @@ extern "C" {
fine DSB() __asm__ volatile("dsb" ::: "memory")
def __aarch64__
fine DMB() __asm__ volatile("DMB ISHLD" : : : "memory")
if defined(__riscv) && (__riscv_xlen == 64)
fine DMB() __asm__ volatile("fence " "r" "," "r" : : : "memory"
se
fine DMB() __asm__ volatile("dmb" ::: "memory")
dif
```

OpenHarmony软件栈RISC-V移植举例

- **编译选项和配置**

```
--- a/musl_template.gni
+++ b/musl_template.gni
@@ -106,6 +106,8 @@ template("musl_libs") {
     if (musl_arch == "arm") {
       cflags_basic += [ "-mtp=cp15" ]
     } else if (musl_arch == "aarch64") {
+    } else if (musl_arch == "riscv64") {
+      cflags_basic += [ "-mno-relax" ]
     }

     cflags_auto = [
@@ -269,6 +271,8 @@ template("musl_libs") {
       ]
     } else if (musl_arch == "x86_64") {
       cflags += [ "-march=x86-64" ]
+    } else if (musl_arch == "riscv64") {
+      cflags += [ "-march=rv64imafdc" ]
     }


@@ -384,7 +388,7 @@ template("musl_libs") {
         "src/math/pow.c",
         "src/math/powf.c",
       ]
-    } else if (musl_arch == "x86_64") {
+    } else if (musl_arch == "x86_64" || musl_arch == "riscv64") {
       sources_orig -= [ "src/thread/${musl_arch}/__set_thread_area.s" ]
     }
```

- **寄存器**
  - 参考内核源码，涉及寄存器的定义
  - 查RISC-V寄存器表格，确定序号

```
+#include "dfx_define.h"
+#include "dfx_logger.h"
+
+namespace OHOS {
+namespace HiviewDFX {
+enum RegisterSeqNum {
+    REG_RISCV64_X0 = 0,
+    REG_RISCV64_X1,
+    REG_RISCV64_X2,
+    REG_RISCV64_X3,
+    REG_RISCV64_X4,
+    REG_RISCV64_X5,
+    REG_RISCV64_X6,
+    REG_RISCV64_X7,
+    REG_RISCV64_X8,
+    REG_RISCV64_X9,
```

```
+++ b/common/dfx_define.h
@@ -45,6 +45,11 @@ static const int REG_SP_NUM = 31;
 #elif defined(__x86_64__)
 static const int USER_REG_NUM = 27;
 static const int REG_PC_NUM = 16;
+#elif defined(__riscv) && (__riscv_xlen == 64)
+static const int USER_REG_NUM = 33;
+static const int REG_PC_NUM = 32;
+static const int REG_LR_NUM = 5;//RISCV Register t0
+static const int REG_SP_NUM = 2;
 #endif
```

- 其他：如宏定义、头文件适配等（参照ARM64，难度低）

任务1 – LLVM15 support ark gc

- **实验内容**
  - 理解其他架构适配https://gitee.com/openharmony/third_party_llvm-project/pulls/238/files
  - 完成 llvm/lib/CodeGen/PrologEpilogInserter.cpp的RISC-V适配
  - 完善 llvm/lib/Target/RISC-V 功能
  - test262 aot riscv pass
- **评分标准（折算为百分制）**
  - （10分）完成arkcompiler和llvm环境搭建和源码编译
  - （20分）在x86或ARM64架构上完成ark gc功能测试
  - （30分）完成PrologEpilogInserter.cpp适配并提交pr至riscv-sig
  - （30分）完成llvm/lib/Target/RISCV相关功能并提交pr至riscv-sig
  - （10分）书面实验报告

任务1 – LLVM15 support ark gc（陈）

根据任务内容其它架构的适配进行修改：

llvm/lib/CodeGen/PrologEpilogInserter.cpp:

```
void PEI::RecordCalleeSaveRegisterAndOffset(MachineFunction &MF, const std::vector<CalleeSavedInfo> &CSI)
...
  if ((archType != Triple::aarch64 && archType != Triple::x86_64) || !(TFI->hasFP(MF))) { // add riscv ?
    return;
  }
  unsigned FpRegDwarfNum = 0;
  if (archType == Triple::aarch64) {                    // add riscv register ?
    FpRegDwarfNum = 29; // x29
  } else {
    FpRegDwarfNum = 6; //rbp
  }
…
    if ((DwarfRegNum == LinkRegDwarfNum || DwarfRegNum == FpRegDwarfNum)
      && (archType == Triple::aarch64)) {             // check riscv ?
      continue;
    }
```

任务1 – LLVM15 support ark gc（陈）

根据任务内容其它架构的适配进行修改：

llvm/lib/Target/AArch64/GISel/AArch64CallLowering.cpp

```
static bool doesCalleeRestoreStack(CallingConv::ID CallConv, bool TailCallOpt) {
#ifdef ARK_GC_SUPPORT
  return ((CallConv == CallingConv::GHC || CallConv == CallingConv::Fast) && TailCallOpt) ||
         CallConv == CallingConv::Tail || CallConv == CallingConv::SwiftTail;
#else
  return (CallConv == CallingConv::Fast && TailCallOpt) ||
         CallConv == CallingConv::Tail || CallConv == CallingConv::SwiftTail;
#endif
}
```

任务2 – trampoline跳转桥函数适配

- **任务描述**
  - 理解其它架构实现 https://gitee.com/riscv-sig/arkcompiler_ets_runtime/tree/master/ecmascript/compiler/trampoline/aarch64
  - 编写RISCV对应的接口
- **评分标准（折算为百分制）**
  - （10分）**完成**arkcompiler环境搭建和源码编译
  - （30分）**完成**arm64**相**应函数解析
  - （30分）**完成函数的**riscv实现并提交pr**至**riscv-sig
  - （20分）**能逐行解**释代码实现
  - （10分）书面实验报告

任务2 – trampoline跳转桥函数适配(陈)

比较在aarch64和x64同名函数实现riscv适配：

https://gitee.com/riscv-sig/arkcompiler_ets_runtime/blob/master/ecmascript/compiler/trampoline/aarch64/asm_interpreter_call.cpp#L35

https://gitee.com/riscv-sig/arkcompiler_ets_runtime/blob/master/ecmascript/compiler/trampoline/x64/asm_interpreter_call.cpp#L40

```
void AsmInterpreterCall::AsmInterpreterEntry(ExtendedAssembler *assembler)
{
    __ BindAssemblerStub(RTSTUB_ID(AsmInterpreterEntry));
    Label target;
    size_t begin = __ GetCurrentPosition();
    PushAsmInterpEntryFrame(assembler);
    __ Bl(&target);                                 // change to riscv function
    PopAsmInterpEntryFrame(assembler);
    size_t end = __ GetCurrentPosition();
    if ((end - begin) != FrameCompletionPos::ARM64EntryFrameDuration) {
        LOG_COMPILER(FATAL) << (end - begin) << " != " << FrameCompletionPos::ARM64EntryFrameDuration
                            << "This frame has been modified, and the offset EntryFrameDuration should be updated too.";
    }
    __ Ret();

    __ Bind(&target);
    {
        AsmInterpEntryDispatch(assembler);
    }
}
```

任务2 – trampoline跳转桥函数适配(陈)

比较在aarch64和x64同名函数实现riscv适配：

https://gitee.com/riscv-sig/arkcompiler_ets_runtime/blob/master/ecmascript/compiler/trampoline/aarch64/common_call.cpp#L35

https://gitee.com/riscv-sig/arkcompiler_ets_runtime/blob/master/ecmascript/compiler/trampoline/x64/common_call.cpp#L44

```
void CommonCall::PushAsmInterpBridgeFrame(ExtendedAssembler *assembler)
{
    Register fp(X29);          // change to riscv function
    Register sp(SP);

    [[maybe_unused]] TempRegister1Scope scope1(assembler);
    Register frameTypeRegister = __ TempRegister1();

    __ Mov(frameTypeRegister, Immediate(static_cast<int64_t>(FrameType::ASM_INTERPRETER_BRIDGE_FRAME)));
    // 2 : return addr & frame type
    __ Stp(frameTypeRegister, Register(X30), MemoryOperand(sp, -2 * FRAME_SLOT_SIZE, AddrMode::PREINDEX));
    // 2 : prevSp & pc
    __ Stp(Register(Zero), Register(FP), MemoryOperand(sp, -2 * FRAME_SLOT_SIZE, AddrMode::PREINDEX));
    __ Add(fp, sp, Immediate(24));  // 24: skip frame type, prevSp, pc

    if (!assembler->FromInterpreterHandler()) {
        __ CalleeSave();
    }
}
```

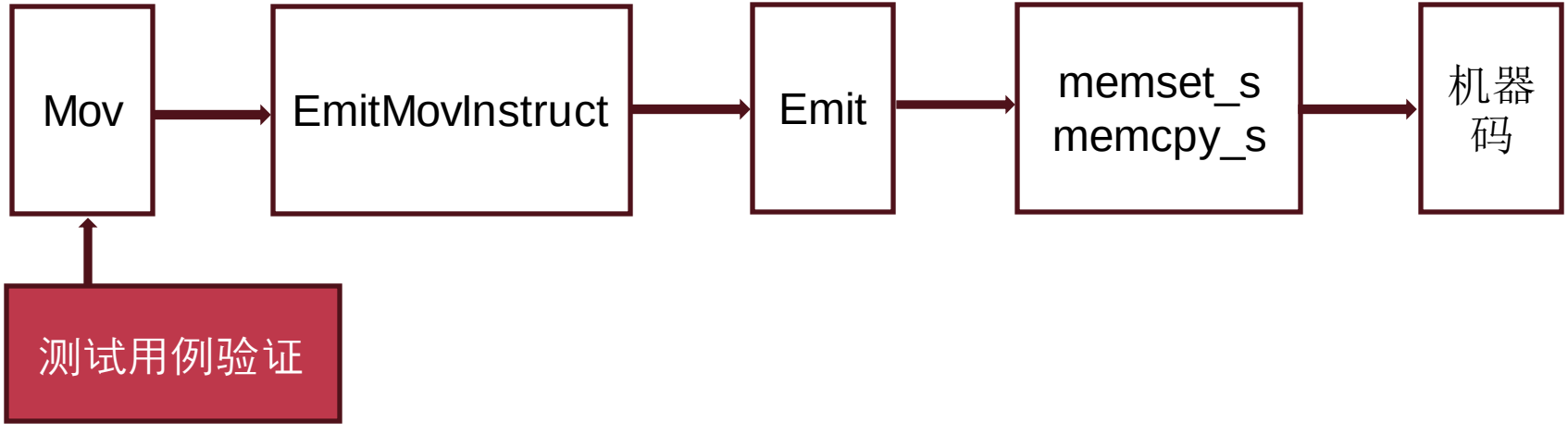
任务3 – ets runtime assembler适配

- **实验内容**
  - 根据tests理解其他架构实现https://gitee.com/riscv-sig/arkcompiler_ets_runtime/tree/master/ecmascript/compiler/assembler/tests
  - 参考V8 Engine完成 Assembler类各个接口的实现 https://gitee.com/mirrors/V8/tree/main/src/codegen/riscv
  - 编写测试用例并通过测试
- **评分标准（折算为百分制）**
  - （10分）完成arkcompiler环境搭建和源码编译
  - （30分）完成Assembler类各个接口的实现并提交pr至riscv-sig
  - （20分）能逐行解释代码实现
  - （30分）完成相应测试用例并提交pr至riscv-sig
  - （10分）书面实验报告

任务3 – ets runtime assembler适配

参考AssemblerAarch64::Mov接口，该接口在return前会调用AssemblerAarch64 ::EmitMovInstruct，最终走到DynChunk::Emit，对变化后的入参进行memset_s 、memcpy_s等操作，最终转向机器码。

在 assembler_aarch64_test.cpp 中有对应AssemblerAarch64::Mov接口的测试用例HWTEST_F_L0(AssemblerAarch64Test, Mov)，用于测试AssemblerAarch64::Mov接口转出的机器码是否正确。



任务3 – ets runtime assembler适配

新建riscv的Assembler类，并实现Mov接口

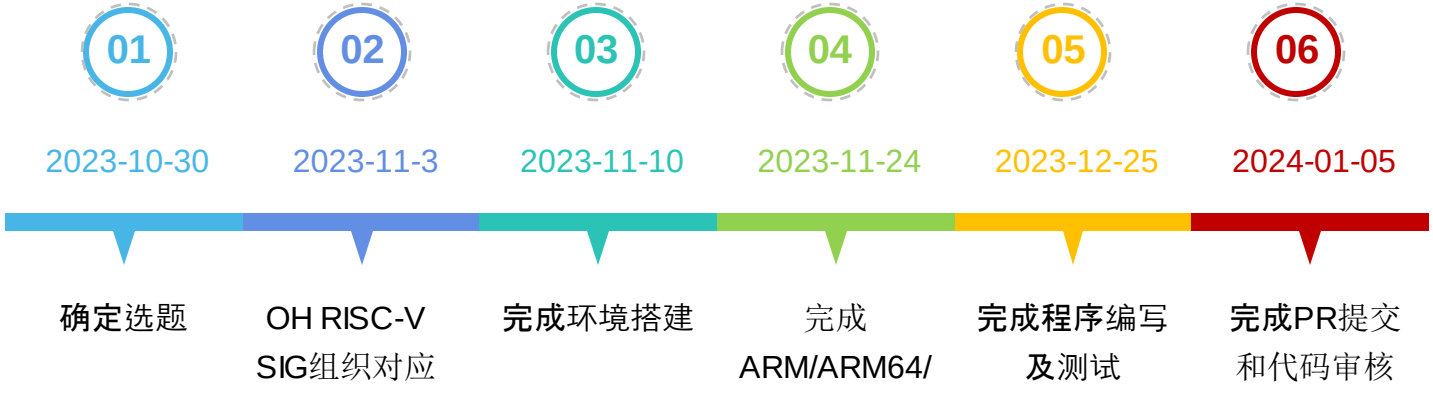
```
void AssemblerAarch64::Mov(const Register &rd, const Immediate &imm)
{
    ASSERT_PRINT(!rd.IsSp(), "sp can't load immediate, please use add instruction");
    const unsigned int HWORDSIZE = 16;
    uint64_t immValue = static_cast<uint64_t>(imm.Value());
    unsigned int allOneHalfWords = 0;
    unsigned int allZeroHalfWords = 0;
    unsigned int regSize = rd.IsW() ? RegWSize : RegXSize;
    unsigned int halfWords = regSize / HWORDSIZE;

    for (unsigned int shift = 0; shift < regSize; shift += HWORDSIZE) { ...
    // use movz/movn over ORR.
    if (((halfWords - allOneHalfWords) <= 1) && ((halfWords - allZeroHalfWords) <= 1)) { ...
    // Try a single ORR.
    uint64_t realImm = immValue << (RegXSize - regSize) >> (RegXSize - regSize);
    LogicalImmediate orrImm = LogicalImmediate::Create(realImm, regSize);
    if (orrImm.IsValid()) { ...
    // One to up three instruction sequence.
    if (allOneHalfWords >= (halfWords - 2) || allZeroHalfWords >= (halfWords - 2)) { ...
    ASSERT_PRINT(regSize == RegXSize, "all 32-bit Immediate will be transformed with a MOVZ/MOVK pair");

    for (unsigned int shift = 0; shift < regSize; shift += HWORDSIZE) { ...

    if (allOneHalfWords || allZeroHalfWords) { ...

    if (regSize == RegXSize && TryReplicateHWords(rd, realImm)) { ...

    if (regSize == RegXSize && TrySequenceOfOnes(rd, realImm)) { ...
    EmitMovInstruct(rd, immValue, allOneHalfWords, allZeroHalfWords);
    return;
} « end Mov »
```

任务3 – ets runtime assembler适配

```
HWTEST_F_L0(AssemblerAarch64Test, Mov)
{
    std::string expectResult("00000000:d28acf01 \tmov\tx1, #22136\n"
                             "00000004:f2a24681 \tmovk\tx1, #4660, lsl #16\n"
                             "00000008:f2ffffe1 \tmovk\tx1, #65535, lsl #48\n"
                             "0000000c:d2801de2 \tmov\tx2, #239\n"
                             "00000010:f2b579a2 \tmovk\tx2, #43981, lsl #16\n"
                             "00000014:f2cacf02 \tmovk\tx2, #22136, lsl #32\n"
                             "00000018:f2e24682 \tmovk\tx2, #4660, lsl #48\n"
                             "0000001c:b2683be3 \tmov\tx3, #549739036672\n"
                             "00000020:f2824683 \tmovk\tx3, #4660\n"
                             "00000024:32083fe4 \tmov\tw4, #-16776961\n");
    AssemblerAarch64 masm(chunk_);
    __ Mov(Register(X1),  Immediate(0xffff000012345678));
    __ Mov(Register(X2),  Immediate(0x12345678abcd00ef));
    __ Mov(Register(X3),  Immediate(0x7fff001234));
    __ Mov(Register(X4).W(),  Immediate(0xff0000ff));
    std::ostringstream oss;
    DisassembleChunk("aarch64-unknown-linux-gnu", &masm, oss);
    ASSERT_EQ(oss.str(), expectResult);
} « end HWTEST_F_L0 »
```

重写测试用例，完成对riscv的Mov接口的测试

Roadmap


01
02
03
04
05
06
2023-10-30
2023-11-3
2023-11-10
2023-11-24
2023-12-25
2024-01-05
确定选题
OH RISC-V SIG组织对应代码仓下完成任务issue创建，分配老师
完成环境搭建
完成ARM/ARM64/X64等一种已有函数解析
完成程序编写及测试
完成PR提交和代码审核

环境需求

系统：Ubuntu版本推荐18.04或20.04

磁盘： 100G

CPU：4 核心、8 线程

内存： 8 Gib 内存

使用指南：
https://gitee.com/openharmony/arkcompiler_ets_runtime/blob/master/docs/README_zh.md

知识产权

代码上官方主线，因版本升级导致
代码适当修改，**保留学生邮箱**

作业老师

https://gitee.com/openharmony

https://gitee.com/riscv-sig

学生

适配代码使用**国科大邮箱**提交pr

每一位参与的同学，补丁会进入OpenHarmony主线，传播到世界各地