单个 sikernel case 生成 arch_model vector dump 使用说明

本文介绍如何使用 `sikernel/scripts/run_one_arch_model_vector_dump_case.sh` 为单个 sikernel case 生成 arch_model vector dump。

适用场景：

- 只想对某一个 sikernel case 生成 vector dump，不想跑完整 sikernel 回归。
- 想快速验证某个 case 在 arch_model 下能否运行。
- 想拿到 `ilm.desc`、`dram_init.desc`、`dram_result.desc` 等 vector dump 文件用于后续架构验证或定位问题。

本文默认脚本已经合入到 `sikernel/scripts/` 目录下。

1. 你需要提前准备什么

1.1 机器环境

建议在公司已有 SiPU 开发环境的 Linux 机器上操作。至少需要：

- `bash`
- `git`
- `python3`
- `python`，需要指向 Python 3，因为脚本内部调用的是 `python`
- `pip`
- `cmake`，建议版本不低于 `3.27`
- 可访问公司 GitLab 的 SSH 权限
- 可访问本地预编译 SDK，默认路径为：

```
代码块
1  /share_data/sicx_sdk/release/latest/sipu_sdk_setup.sh
```

脚本会自动执行：

```
代码块
```

```
source sikernel/set_sdk.sh
```

因此正常情况下不需要你手动编译 `sipu_sw` 里的 SDK，也不需要手动 source SDK。只要 `sikernel/set_sdk.sh` 指向的 SDK 可用即可。

如果你是在一台新机器上准备环境，可以先安装基础工具。Ubuntu / Debian 系统可以使用：

代码块

```
sudo apt-get update
sudo apt-get install -y \
  git \
  openssh-client \
  python3 \
  python3-pip \
  python3-venv \
  python-is-python3 \
  cmake \
  build-essential
```

如果你没有 `sudo` 权限，联系机器管理员安装上面的系统工具；Python 包可以安装到自己的用户目录或虚拟环境里。

安装后检查：

代码块

```
bash --version | head -1
git --version
python --version
python3 --version
python3 -m pip --version
cmake --version
```

注意：

- `python --version` 必须显示 Python 3.x。
- 如果 `python` 命令不存在，脚本里的 Python 调用会失败。可以安装 `python-is-python3`，或者使用 Python 虚拟环境。
- 如果 `cmake --version` 低于 `3.27`，建议参考后面的 CMake 安装方式升级。

1.2 检查 GitLab SSH 权限

从空白目录开始之前，可以先确认自己能访问 GitLab：

代码块

```
ssh -T git@gitlabsoft.siorigin.com
```

如果提示没有权限、连接失败、需要密码，先配置 SSH key 或联系仓库管理员开通权限。

1.3 检查 SDK 是否存在

代码块

```
ls -l /share_data/sicx_sdk/release/latest/sipu_sdk_setup.sh
```

如果这个文件不存在，脚本后面会在 source SDK 时失败。需要先确认机器上是否安装了 SDK，或者让维护者更新 `sikernel/set_sdk.sh` 指向正确 SDK。

1.4 CMake 版本不足时怎么安装

部分系统自带的 CMake 版本可能比较低。如果 `cmake --version` 显示低于 `3.27`，可以安装用户态 CMake：

代码块

```
python3 -m pip install --user -U cmake
export PATH="$HOME/.local/bin:$PATH"
cmake --version
```

如果你使用虚拟环境，可以在虚拟环境里安装：

代码块

```
python -m pip install -U cmake
cmake --version
```

为了下次登录仍然能找到用户态 CMake，可以把下面这行加入 `~/.bashrc`：

代码块

```
export PATH="$HOME/.local/bin:$PATH"
```

1.5 推荐使用 Python 虚拟环境

新手建议使用虚拟环境，这样不会污染系统 Python，也能保证脚本里调用的 `python` 就是 Python 3。

代码块

```
python3 -m venv ~/.venvs/sipu-vector-dump
source ~/.venvs/sipu-vector-dump/bin/activate
python --version
python -m pip install -U pip
```

之后同一个终端里继续执行本文后续命令即可。

如果你重新打开了一个终端，需要再次激活：

代码块

```
source ~/.venvs/sipu-vector-dump/bin/activate
```

2. 从空白目录拉取代码

下面假设你从一个干净目录开始：

代码块

```
mkdir -p ~/workspace/sipu_vector_dump
cd ~/workspace/sipu_vector_dump
```

2.1 推荐方式：完整拉取 sipu_sw1.5 及子仓

这是对新手最省心的方式，缺点是拉取内容较多、占用空间较大。

代码块

```
git clone git@gitlabsoft.siorigin.com:arch/sipu_sw1.5.git
cd sipu_sw1.5
git submodule update --init --recursive
```

如果你已经 clone 过仓库，只需要更新：

代码块

```
cd ~/workspace/sipu_vector_dump/sipu_sw1.5
git pull
git submodule update --init --recursive
```

2.2 省空间方式：只拉脚本必需的子仓

如果你只是想使用这个单 case vector dump 脚本，通常只需要初始化三个子仓：

代码块

```
git clone git@gitlabsoft.siorigin.com:arch/sipu_sw1.5.git
cd sipu_sw1.5
git submodule update --init sikernel ci_test arch_test_tools
```

注意：

- 这个方式更省空间。
- 但某些 sikernel case 可能依赖 `sikernel` 内部的其他子模块，例如 `siblas`、`sipu_libm`、`sidnn` 等。
- 如果遇到编译失败，且错误指向缺少这些目录，再按需补拉对应子模块。

2.3 确认脚本版本

clone 完成后，先确认当前仓库里的脚本已经是本文描述的版本：

代码块

```
cd ~/workspace/sipu_vector_dump/sipu_sw1.5
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh --help | grep -E 'build-and-run|VECTOR DUMP|On success' || true
```

正常情况下至少应该能看到：

代码块

```
On success, print the generated vector_dump directory.
--build-and-run
```

如果看不到 `--build-and-run`，说明你当前 checkout 的 `sikernel` 子仓还不是包含该脚本改动的版本。先更新到已合入该脚本的分支或提交，再继续后面的步骤。

3. 安装 Python 依赖

3.1 完整依赖安装

如果你完整拉取了所有子仓，可以直接使用仓库提供的依赖安装脚本：

代码块

```
cd ~/workspace/sipu_vector_dump/sipu_sw1.5
bash install_dependencies.sh
```

这个脚本会安装 Python 依赖，并构建部分工具。第一次执行可能比较慢。

注意：

- 如果你已经激活了虚拟环境，依赖会安装到虚拟环境里。
- 如果你没有激活虚拟环境，依赖会安装到当前默认 Python 环境里。
- 如果遇到权限错误，优先使用虚拟环境；不建议直接用 `sudo pip install`。

3.2 只安装本脚本常用依赖

如果你不想执行完整安装脚本，可以至少安装下面这些 Python 包：

代码块

```
python3 -m pip install --user \
  pytest \
  pytest-xdist \
  pytest-json-report \
  pytest-html \
  pytest-timeout \
  pytest-timestamper \
  tomlkit \
  Jinja2
```

3.3 推荐安装方式：虚拟环境内安装

如果你按照前面建议创建了虚拟环境，推荐使用下面这组命令：

代码块

```
source ~/.venvs/sipu-vector-dump/bin/activate
python -m pip install -U pip
python -m pip install \
  pytest \
  pytest-xdist \
  pytest-json-report \
  pytest-html \
  pytest-timeout \
```

```
    pytest-timestamper \
    tomlkit \
    Jinja2
```

安装后验证：

代码块

```
python - <<'PY'
import pytest
import tomlkit
import jinja2
print("python deps ok")
PY

python -m pytest --version
python -m pytest --help | grep -E 'json-report|html|numprocesses|dist' || true
```

如果 `pytest --help` 里能看到 `--json-report`、`--html`、`-n` 或 `--dist` 相关选项，说明插件基本可用。

3.4 依赖清单说明

依赖	安装方式	用途
`git`	`sudo apt-get install git`	拉取 `sipu_sw1.5` 和子仓。
`openssh-client`	`sudo apt-get install openssh-client`	通过 SSH 访问公司 GitLab。
`python3` / `python`	`sudo apt-get install python3 python-is-python3` 或虚拟环境	脚本和测试框架都需要 Python，脚本内部使用 `python` 命令。
`pip`	`sudo apt-get install python3-pip` 或虚拟环境自带	安装 Python 包。
`cmake >= 3.27`	系统安装或 `python -m pip install -U cmake`	编译单个 sikernel case。
`build-essential`	`sudo apt-get install build-essential`	提供基础 C/C++ 编译工具。
`pytest`	`python -m pip install pytest`	运行生成的 regression pytest。

`pytest-xdist`	`python -m pip install pytest-xdist`	支持 `-n`、`--dist` 参数。
`pytest-json-report`	`python -m pip install pytest-json-report`	生成 `report.json`。
`pytest-html`	`python -m pip install pytest-html`	生成 `report.html`。
`tomlkit`	`python -m pip install tomlkit`	读写 `archmodel.toml`。
`Jinja2`	`python -m pip install Jinja2`	SDK setup 和部分工具依赖。
SiPU SDK	由机器预装，检查 `/share_data/sicx_sdk/release/latest/sipu_sdk_setup.sh`	提供 `scc`、`archmodel.toml`、`libsipu.so`、`libsipurt.so` 等。

如果你的环境使用 conda 或虚拟环境，可以去掉 `--user`：

代码块

```
python3 -m pip install \
  pytest \
  pytest-xdist \
  pytest-json-report \
  pytest-html \
  pytest-timeout \
  pytest-timestamper \
  tomlkit \
  Jinja2
```

4. 验证环境是否准备好

进入 `sipu_sw1.5` 根目录：

代码块

```
cd ~/workspace/sipu_vector_dump/sipu_sw1.5
```

检查脚本是否存在：

代码块

```
ls -l sikernel/scripts/run_one_arch_model_vector_dump_case.sh
```

查看帮助：

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh --help
```

你应该能看到类似选项：

代码块

```
--build-and-run
--build-only
--run-only
--repo-build
--regression-root
--timeout
```

如果输出前面带有很多 `+`、`++` 开头的调试信息，这是脚本开启了 shell xtrace 的正常现象。只要最后出现 usage 和 options，就说明帮助信息正常打印。

检查 SDK：

代码块

```
source sikernel/set_sdk.sh
echo "${SI_SDK_ROOT}"
which scc
ls -l "${SI_SDK_ROOT}/lib/archmodel.toml"
ls -l "${SI_SDK_ROOT}/lib/libsipu.so"
ls -l "${SI_SDK_ROOT}/lib/libsipurt.so"
```

正常情况下应该能看到：

- `SI_SDK_ROOT` 指向某个 `/share_data/sicx_sdk/release/...` 目录
- `scc` 在 SDK 的 `bin/` 目录下
- `archmodel.toml` 存在
- `libsipu.so` 和 `libsipurt.so` 存在

5. 第一次运行：用 sigmoid 做 smoke test

建议第一次先跑一个较小的 case，确认整体流程没有问题。

在 `sipu_sw1.5` 根目录执行：

```
代码块
1  bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
2    --build-and-run \
3    source/source_builtin/attention/sigmoid
```

`--build-and-run` 表示：

1. 复制一份 `sikernel` 到 build 目录。
2. 只 build 你指定的 case。
3. 基于 build 结果创建 regression 目录。
4. 用 `arch_model` 跑这个 case。
5. 开启 vector dump。
6. 检查并打印生成的 vector dump 目录。

成功时最后会看到类似输出：

```
代码块
1  VECTOR DUMP SUCCESS:
   /path/to/sipu_sw1.5/sikernel_arch_model_vector_dump_sigmoid/source__source_buil
   tin__attention__sigmoid/vector_dump (61 files)
```

只要看到 `VECTOR DUMP SUCCESS`，说明 vector dump 已经生成成功。

耗时参考：在一台已有 SDK 和 Python 依赖的开发机上，`sigmoid` smoke test 的 build 大约 40 秒，arch_model vector dump run 大约 7 秒。不同机器会有差异；如果卡住很久，可以先等到 `--timeout` 设置的时间，再根据报错排查。

6. 生成结果在哪里

如果不指定 `--regression-root`，脚本默认会在 `sipu_sw1.5` 根目录下生成：

```
代码块
1  sikernel_arch_model_vector_dump_<case最后一级目录名>/
```

例如：

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  source/source_builtin/attention/sigmoid
```

默认输出目录类似：

代码块

```
sipu_sw1.5/
└── sikernel_arch_model_vector_dump_sigmoid/
    └── source__source_builtin__attention__sigmoid/
        └── vector_dump/
            ├── ilm.desc
            ├── dram_init.desc
            ├── dram_result.desc
            ├── ilm0.bin
            ├── dram_init0.bin
            ├── dram_result0.bin
            └── ...
```

脚本会检查下面三个文件是否存在且非空：

- `ilm.desc`
- `dram_init.desc`
- `dram_result.desc`

如果这些文件缺失或为空，脚本会认为 vector dump 失败。

7. 推荐使用方式：显式指定输出目录

多人共用机器时，建议把 build 目录和 regression 目录放到 `.tmp/` 或自己的工作目录里，避免互相覆盖。

代码块

```
cd ~/workspace/sipu_vector_dump/sipu_sw1.5
mkdir -p .tmp

bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --repo-build .tmp/sigmoid_build \
  --regression-root .tmp/sigmoid_arch_model_vector_dump \
  --build-and-run \
  source/source_builtin/attention/sigmoid
```

成功后结果在：

代码块

```
.tmp/sigmoid_arch_model_vector_dump/source__source_builtin__attention__sigmoid/vector_dump/
```

8. 跑你自己的 case

脚本参数里的 `case_path` 是相对于 `sikernel` 根目录的路径。

例如 case 实际目录是：

代码块

```
sipu_sw1.5/sikernel/source/source_builtin/attention/sigmoid
```

那么传给脚本的是：

代码块

```
source/source_builtin/attention/sigmoid
```

也可以带 `sikernel/` 前缀，脚本会自动去掉：

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  sikernel/source/source_builtin/attention/sigmoid
```

8.1 查找可用 case

可以用下面命令找带 `build.sh` 的 case：

代码块

```
find sikernel/source/source_builtin -name build.sh \
  | sed 's#^sikernel/##; s#/build.sh$##' \
  | sort \
  | head -50
```

拿到路径后，直接传给脚本即可。

9. 常用命令

9.1 build + run，一步完成

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --build-and-run \
  <case_path>
```

`--build-and-run` 是显式写法。不写这个选项时，默认行为也是 build + run。

9.2 只 build，不 run

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --repo-build .tmp/my_case_build \
  --build-only \
  <case_path>
```

适合先确认 case 能不能编译。

9.3 已经 build 过，只重新 run vector dump

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --repo-build .tmp/my_case_build \
  --regression-root .tmp/my_case_vector_dump \
  --run-only \
  <case_path>
```

注意：

- `--run-only` 要求 `--repo-build` 指向已经 build 好的目录。
- 如果你改了 kernel 或测试代码，需要重新 build。

9.4 修改超时时间

默认超时时间是 480 秒。如果 case 比较慢，可以加大：

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --timeout 1200 \
  --build-and-run \
```

```
4      <case_path>
```

10. 参数说明

参数	说明
`--build-and-run`	build 指定 case，然后运行 arch_model vector dump。默认行为也是这个。
`--build-only`	只 build，不运行 vector dump。
`--run-only`	只运行 vector dump，要求 build 目录已经存在。
`--repo-build <dir>`	build 用的 sikernel 副本目录。默认是 `sikernel_build_one_case`。
`--regression-root <dir>`	regression 和 vector dump 输出目录。默认按 case 名自动生成。
`--timeout <sec>`	运行超时时间，单位秒，默认 480。
`-h` / `--help`	打印帮助信息。

`--build-and-run`、`--build-only`、`--run-only` 三者互斥，不能同时使用。

11. 结果文件怎么看

最重要的目录是：

代码块

```
1   <regression-root>/<case_name>/vector_dump/
```

其中 `<case_name>` 会把路径里的 `/` 替换成 `__`，把 `-` 和 `.` 替换成 `_`。

例如：

代码块

```
1   source/source_builtin/attention/sigmoid
```

会变成：

```
source__source_builtin__attention__sigmoid
```

常见文件：

文件	含义
`ilm.desc`	指令/ILM 初始化相关描述。
`dram_init.desc`	输入 DRAM 初始化数据描述。
`dram_result.desc`	输出 DRAM 结果数据描述。
`*.bin`	由 raw vector 切分出的二进制片段。
`scope_pe` / `scope_pec` / `scope_peg` / `scope_chip`	workload 作用范围标记文件，实际生成哪些取决于 case。

12. 清理临时目录

如果你使用默认目录，清理示例：

代码块

```
rm -rf sikernel_build_one_case
rm -rf sikernel_arch_model_vector_dump_*
```

如果你使用 `.tmp/`：

代码块

```
rm -rf .tmp
```

确认目录没有重要结果后再删除。

13. 常见问题

13.0 clone 或 git submodule 时出现 locale 警告

如果看到类似警告：

代码块

```
sh: warning: setlocale: LC_ALL: cannot change locale (C.UTF-8)
```

这通常不影响 clone 和 submodule 初始化，可以先继续后面的步骤。如果你想消除这个警告，可以先查看当前机器支持哪些 locale：

```
代码块
1   locale -a
```

然后选择一个存在的 locale，例如：

```
代码块
1   export LC_ALL=C
2   export LANG=C
```

或者如果机器支持 `C.UTF-8`：

```
代码块
1   export LC_ALL=C.UTF-8
2   export LANG=C.UTF-8
```

13.1 找不到仓库根目录

报错类似：

```
代码块
1   can't locate repo root from script path
```

说明脚本没有放在正确的 `sipu_sw1.5/sikernel/scripts/` 结构下，或者 `ci_test` 子仓没有初始化。

处理方式：

```
代码块
1   cd sipu_sw1.5
2   git submodule update --init sikernel ci_test arch_test_tools
```

13.2 找不到 arch_test_tools

报错类似：

代码块

```
can't find arch_test_tools python package
```

处理方式：

代码块

```
cd sipu_sw1.5
git submodule update --init arch_test_tools
```

13.3 找不到 SDK

报错可能来自：

代码块

```
/share_data/sicx_sdk/release/latest/sipu_sdk_setup.sh: No such file or
directory
```

处理方式：

代码块

```
ls -l /share_data/sicx_sdk/release/latest/sipu_sdk_setup.sh
```

如果文件不存在，需要在当前机器准备 SDK，或者让维护者修改 `sikernel/set_sdk.sh` 指到正确路径。

13.4 找不到 scc 或 CMake 找不到 scc package

报错可能类似：

代码块

```
Could not find a package configuration file provided by "scc"
```

先检查 SDK 环境：

代码块

```
cd sipu_sw1.5
source sikernel/set_sdk.sh
```

```
which scc
echo "${CMAKE_PREFIX_PATH}"
```

如果 `which scc` 没有输出，说明 SDK 没有正确加载。

13.5 Python 缺包

报错类似：

代码块

```
ModuleNotFoundError: No module named 'tomlkit'
ModuleNotFoundError: No module named 'pytest'
```

处理方式：

代码块

```
python3 -m pip install --user \
  pytest \
  pytest-xdist \
  pytest-json-report \
  pytest-html \
  pytest-timeout \
  pytest-timestamper \
  tomlkit \
  Jinja2
```

13.6 pytest 不认识 json-report 或 html 参数

报错类似：

代码块

```
unrecognized arguments: --json-report
unrecognized arguments: --html
```

说明 pytest 插件没有装：

代码块

```
python3 -m pip install --user pytest-json-report pytest-html pytest-xdist
```

13.7 build 失败

先单独 build：

代码块

```
bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --repo-build .tmp/debug_build \
  --build-only \
  <case_path>
```

如果这里失败，问题通常在 case 自己的 `build.sh`、`CMakeLists.txt`、源码 include、SDK 编译工具链或 case 依赖子模块。

13.8 run 成功但没有 VECTOR DUMP SUCCESS

脚本只有在找到非空的：

- `ilm.desc`
- `dram_init.desc`
- `dram_result.desc`

后才会打印 `VECTOR DUMP SUCCESS`。

如果没有打印，说明 vector dump 目录没有生成完整。可以检查：

代码块

```
find <regression-root> -maxdepth 4 -type d -name vector_dump
find <regression-root> -maxdepth 5 -type f | sort | head -100
```

同时确认运行时使用的是 `arch_model`，并且 SDK 里的 `archmodel.toml` 存在。

14. 一条完整示例

下面是一条从空白目录开始的完整示例：

代码块

```
mkdir -p ~/workspace/sipu_vector_dump
cd ~/workspace/sipu_vector_dump

git clone git@gitlabsoft.siorigin.com:arch/sipu_sw1.5.git
cd sipu_sw1.5

git submodule update --init --recursive

```

```
python3 -m pip install --user \
  pytest \
  pytest-xdist \
  pytest-json-report \
  pytest-html \
  pytest-timeout \
  pytest-timestamper \
  tomlkit \
  Jinja2

source sikernel/set_sdk.sh
which scc
ls -l "${SI_SDK_ROOT}/lib/archmodel.toml"

mkdir -p .tmp

bash sikernel/scripts/run_one_arch_model_vector_dump_case.sh \
  --repo-build .tmp/sigmoid_build \
  --regression-root .tmp/sigmoid_arch_model_vector_dump \
  --build-and-run \
  source/source_builtin/attention/sigmoid
```

看到下面这行就说明成功：

代码块

```
VECTOR DUMP SUCCESS: .../vector_dump (... files)
```

结果目录：

代码块

```
.tmp/sigmoid_arch_model_vector_dump/source__source_builtin__attention__sigmoid/
vector_dump/
```