

周易 CompassStudio

3.0 版

用户指南

保密

版权所有 © 2022–2023 安谋科技（中国）有限公司。保留所有权利。

发行号 06

61010019_0300_06_zh

周易 CompassStudio

用户指南

版权所有 © 2022–2023 安谋科技（中国）有限公司。保留所有权利。

版本信息

文档历史

发行号	日期	保密性	变更
0100-01	2022-06-30	保密	1.0 版
0101-02	2022-09-30	保密	1.1 版
0200-03	2022-12-31	保密	2.0 版
0201-04	2023-03-31	保密	2.1 版
0202-05	2023-06-30	保密	2.2 版
0300-06	2023-09-30	保密	3.0 版

专有权声明

本文档为保密资料，您在使用时须遵守您与安谋科技（中国）有限公司（“安谋科技”）之间所签协议的各项条款，以及您与获得安谋科技授权向您披露本文档的有关方之间所签协议的各项条款。

本文档受版权和其他相关权利的保护，实践或实施本文档中所含信息可能受到一项或多项专利或待定专利申请的保护。未经安谋科技事先明确书面许可，不得以任何形式通过任何手段复制本文档的任何部分。除非明确说明，否则本文档未以禁止反言或其他方式授予任何知识产权方面的许可，无论是明示还是暗示许可。

您对本文档所含信息的访问取决于您是否接受不出于以下目的使用或允许他人使用此信息的条件：（1）为了确定实施是否会侵犯任何第三方专利；（2）为了开发技术或产品以避开安谋科技的任何知识产权；或（3）作为修改现有专利或专利申请或制造任何延续、部分延续或扩展现有专利或专利申请的参考；或（4）为了生成数据以发布或披露给第三方，在未获得安谋科技事先书面同意的情况下，将本文档中所述的安谋科技技术的性能或功能性与您或第三方创建的任何其他产品进行比较。

本文档“按原样”提供。安谋科技就本文档不作任何陈述和保证，无论是明示、暗示或法定保证，包括但不限于对适销性、满意的质量、不侵权或针对特定目的的适用性的暗示保证。为避免疑义，安谋科技不就在分析的基础上识别或理解第三方专利、版权、商业机密或其他权利的范围和内容作出任何陈述和承诺。

本文档可能包含技术或排版上的错误。

在不受法律禁止的情况下，安谋科技在任何情况下都不会对因使用本文档引起的任何损害承担责任，包括但不限于任何直接、间接、特殊、连带、惩罚性或后果性损害，无论损害如何造成以及何种责任理论，即便安谋科技已被告知有此类损害的可能性。

本文档仅包含商业内容。您应负责确保本文档的任何使用、复制或披露完全符合任何相关的出口法律法规，以确保本文档或其任何部分不会在违反相关出口法律法规的情况下直接或间接出口。在提及安谋科技的客户时使用“合作伙伴”一词并非意欲建立或指代与任何其他公司的任何合作伙伴关系。安谋科技可以随时对本文档进行更改，恕不另行通知。

如果这些条款中的任何规定与同安谋科技达成的涵盖本文档的任何点击或签署的书面协议中的任何规定有冲突，则以点击或签署的书面协议为准，并取代这些条款中的冲突规定。

本文档可以翻译成其他语言以方便使用，您并且同意，若本文档的英文版与任何翻译版出现任何冲突时，将以协议的英文版条款为准。

带有® 或™ 标记的文字为安谋科技或其分公司在中华人民共和国或其他地方的注册商标或商标。保留所有权利。本文档中提及的其他品牌和名称可能是其各自所有者的商标。

版权所有 © 2022–2023 安谋科技。保留所有权利。

保密状态

本文档是保密文档。本文档只能按照安谋科技和安谋科技交付本文档的接收方所签订的协议条款来使用和分发。

产品状态

本文档中的信息是最终版，即用于开发完成的产品。

网址

www.armchina.com

目录

1 前言	7
1.1 约定	7
1.2 参考文档	8
1.3 反馈	8
2 CompassStudio	10
2.1 CompassStudio 简介	10
2.2 CompassStudio 系统运行环境	11
2.3 CompassStudio 开发环境	12
3 模型工程的使用	19
3.1 创建工程	19
3.1.1 使用模板自动创建项目	19
3.1.2 从已有项目直接导入创建项目	23
3.2 NN-Model 配置参数设置	24
3.2.1 手动编辑 cfg 文本	24
3.2.2 通过 NN-Model 配置界面进行编辑	25
4 Model Build 和 Run	32
4.1 前处理	33
4.2 NN-Model Build	34
4.3 Run on Hardware	36
4.3.1 Run on Hardware 环境配置	36
4.4 Run on Simulator	40
4.5 后处理	42
4.6 Profiler	43
4.6.1 Simulator Profiler	44
4.6.2 Hardware Profiler	46

4.6.3 Multi-core profiler	47
4.7 高级功能.....	50
4.7.1 Compass IR Run 和 Build.....	50
4.7.2 QuantTuning.....	59
4.7.3 Compass IR Visual Editor.....	64
5 算子工程的使用.....	81
5.1 创建算子工程.....	81
5.1.1 使用模板自动创建项目.....	81
5.1.2 从已有项目直接导入创建项目	87
5.1.3 手动创建项目	89
5.1.4 基于已有的 Makefile 创建项目	90
5.2 配置算子工程编译参数.....	93
5.2.1 配置 Compass C 算子工程编译参数	93
5.2.2 配置 Compass OpenCL 算子工程编译参数	104
6 算子工程的编译及部署.....	111
6.1 编译算子工程.....	111
6.1.1 编译 Debug 版本的算子	112
6.1.2 编译 Release 版本的算子	116
6.1.3 编译生成汇编文件	119
6.2 编写算子 plugin.....	122
6.3 部署算子工程.....	123
6.3.1 部署到指定的模型工程.....	123
6.3.2 部署到系统环境	127
6.3.3 算子 Plugin 校验.....	128
7 算子工程的调试.....	130
7.1 Debug on Simulator.....	130
7.1.1 Compass C Debug on Simulator	131
7.1.2 Compass OpenCL Debug on Simulator	137

7.2 Debug on Hardware.....	150
7.2.1 通过网络进行调试.....	151
7.2.2 不通过网络进行调试.....	152
7.3 查看 Variables.....	153
7.4 查看 Registers.....	154
7.5 查看 Memory.....	155
7.6 查看 Disassembly.....	157
8 其他工具.....	158
8.1 使用 Python	158
8.2 在算子工程中增加 Python 语法.....	161
8.3 使用 Terminal	162
9 常见问题.....	166
9.1 如何安装 Qt 和 Graphviz.....	166
9.2 如何设置 Compass SDK 运行环境	166
9.3 CompssStudio 无法打开，报“缺少 SWT 相关库”错误.....	166
9.4 如何在 CompassStudio 中配置 Anaconda 虚拟环境.....	167
9.5 新建工程时，可能会报语法错误	167
9.6 新增 Include 路径出现缓存	167
9.7 算子工程编译出错	168
9.8 找不到设置页面栏	168

1 前言

1.1 约定

以下小节介绍安谋科技公司文档所使用的排版约定。

排版约定

约定	使用说明
斜体	介绍特殊术语，表示交叉参考和引用。
黑体	突出显示界面元素，如菜单名称。表示信号名称。在适当情况下，也用于描述性列表中的术语。
等宽字体	表示可以从键盘输入的文本，如命令、文件和程序名以及源代码。
等宽字体黑体	表示在示例代码以外使用的语言关键字。
等宽字体	表示命令或选项的缩写。可输入带下划线的文本用于代替完整命令或选项名称。
<and>	包含在代码或代码片段中出现的汇编语法的可替换项。例如: MRC p15, 0, <Rd>, <CRn>, <CRm>, <Opcode_2>
小型大写字母	在正文本中用于表示具有特定技术含义的一些术语，这些术语已在 Arm® 词汇表中进行定义。例如，IMPLEMENTATION DEFINED、IMPLEMENTATION SPECIFIC、UNKNOWN 和 UNPREDICTABLE。
 Caution	代表一项建议，如果不遵循，可能会导致系统故障或损坏。
 Warning	代表对系统的一项要求，如果不遵守，可能会导致系统故障或损坏。
 Danger	代表对系统的一项要求，如果不遵守，将导致系统故障或损坏。
 Note	代表一个需要您注意的重要信息。
 Tip	代表一个有用的提示，可能会使执行一项任务变得更容易、更好或更快。
 Remember	代表一个对与你正在阅读的信息有关的重要事项的提醒。

1.2 参考文档

本文档包含针对本产品的信息。有关本产品的其他信息，请参阅以下文档：

表 1-2: 安谋科技公司出版物

文档名称	文档编号	仅限被许可人使用
《Arm China Zhouyi Compass Software Technical Overview》	61010011_0303_00	是
《Zhouyi Compass IR Definition Application Note》	ACN-61010013-010	否
《周易 AIPU 量化算法白皮书》	ACN-AWP-1002A	否

表 1-3: 其他出版物

文档编号	文档名称
-	-

1.3 反馈

安谋科技公司欢迎用户就本产品及其文档提供反馈意见。

关于本产品的反馈

如果您有关于本产品的任何意见或建议，请与您的供应商联系并提供：

- 产品名称。
- 产品序列号或版本。
- 尽可能详细地提供信息加以解释。包括症状和诊断程序（如果有）。

关于本文档的反馈

关于如何对文档内容提供反馈的信息。

如果您发现本文档有任何错误或遗漏之处，请发送电子邮件至 errata@armchina.com，并提供：

- 文档标题。
- 文档编号。
- 您有反馈意见的相关页码（如适用）。
- 您的意见的简单说明。

安谋科技还欢迎您对需要增加和改进之处提出建议。



安谋科技仅在 Adobe Acrobat 和 Acrobat Reader 中进行 PDF 的测试，与任何其他 PDF 阅读器一起使用时，不能保证所示文档的质量。

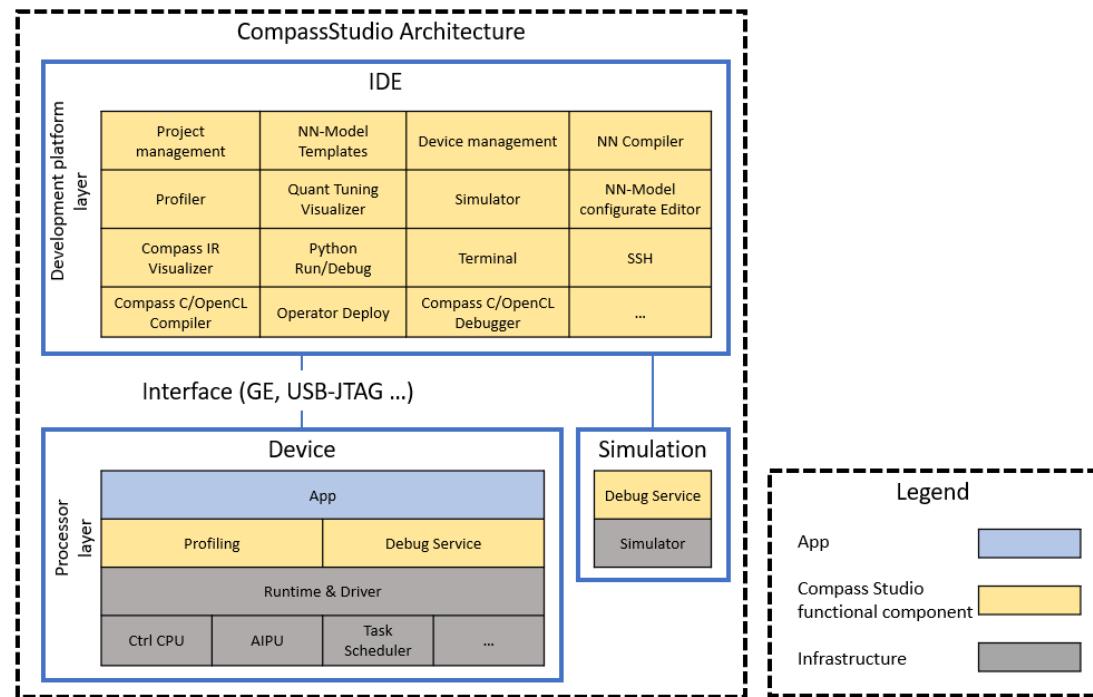
2 CompassStudio

本章节主要介绍 CompassStudio 及其系统运行环境和开发环境。

2.1 CompassStudio 简介

CompassStudio 是基于 Eclipse 平台的一款服务于 Zhouyi Compass NPU 工具链的集成开发环境。CompassStudio 的功能框架如图 2-1 所示。CompassStudio 提供包括项目管理、NN Compiler、Simulator、Profiler、Compass IR Visualizer、QuantTuning、Optimizer、Device 管理、远程登录/远程管理及同步、日志管理、Python 开发、Compass C/OpenCL Compiler、Compass C/OpenCL Debugger 和 Operator Deploy 等。

图 2-1: CompassStudio 功能框架



CompassStudio 主要功能及操作如下:

- 工程管理。主要包括项目创建、打开、关闭、删除和项目管理导出、项目目录创建和项目属性设置。有关详细信息，请参考 [3.1 创建工程](#)。
- Compass SDK 集成。Compass SDK 包含 NN-Compiler, Simulator 等相关工具。查看工具版本信息，请参考 [2.3 CompassStudio 开发环境](#)。
- NN-Model 配置文件可视化编辑，请参考 [3.2 NN-Model 配置参数设置](#)。

- NN Compiler, 支持模型的 Build 和 Run, 以及 Compass IR Build 和 Run, 请参考 [4.2 NN-Model Build](#) 和 [4.7.1 Compass IR Run 和 Build](#)。
- 将模型通过 NN Compiler 运行在 AIPU 的硬件上, 请参考 [4.3 Run on Hardware](#)。
- 将模型通过 NN Compiler 运行在 Simulator 上完成推理的过程, 请参考 [4.4 Run on Simulator](#)。
- Profiler。用来获取网络各层性能信息, 请参考 [4.6 Profiler](#)。
- Compass IR Visualizer。该模块是 Compass IR 可视化的展示, 请参考 [4.7.3 Compass IR Visual Editor](#)。
- Quant Tuning。该功能模块主要支持在 Compass IR 阶段, 对 Quant 和 Optimizer 进行调优展示, 并支持针对每个网络层快捷进入 CompassIR Visualization 视图。详细信息请参考 [4.7.2 QuantTuning](#)。
- Compass C/OpenCL Compiler。该模块主要支持 Compass C 和 Compass OpenCL 的语法解析, 编译等, 请参考 [6.1 编译算子工程](#)。
- Operator Deploy。该模块主要支持 Operator 的部署, 包含 Compass C 和 Compass OpenCL 算子的部署, 请参考 [6.3 部署算子工程](#)。
- Compass C/OpenCL Debugger。该模块主要支持 Compass C 和 Compass OpenCL 的调试, 请参考 [7 算子工程的调试](#)。

目前, CompassStudio 已经充分与 Compass SDK 整合, 可以方便快捷地创建模板工程, 快速修改工程配置选项, 并可根据模板需求自动添加。

2.2 CompassStudio 系统运行环境

解压缩 CompassStudio 软件包 CompassStudio_3.0.0_linux-x86_64.tar.gz。软件包中包含了 CompassStudio 软件。解压路径不可包含中文。

CompassStudio 运行环境

需根据系统 bash 或者 csh 自行设置。本文档中的运行环境以 csh 为准。

- 该版本不需安装 JDK 的环境, CompassStudio 自带 JRE 环境。
- 安装并配置 Compass SDK 依赖, 例如 libc 版本 (有关详细信息, 请参考 [9 常见问题](#))。

```
setenv LD_LIBRARY_PATH $GCC_INSTALL_PATH/lib64:$LD_LIBRARY_PATH
```
- 安装 Compass IR Visual Editor 所依赖的 lib 库。
 - QT 5.9.1: 下载地址为 <https://download.qt.io/archive/qt/5.9/5.9.1/qt-5.9.1.tar.xz>

[opensource-linux-x64-5.9.1.run](#)。请将安装好的 qt 加入系统变量 (LD_LIBRARY_PATH)。详细信息请参考 <https://doc.qt.io/qt-5/linux.html>。

- Graphviz: 下载地址为 <https://graphviz.org/download/>。
- CompassStudio 运行的系统环境 (可选)
配置该系统环境, 无需打开 CompassStudio 安装目录即可运行 CompassStudio。

```
setenv PATH COMPASSSTUDIO_INSTALL_PATH:$PATH
```

目前已在 Linux Redhat_7.5.x_64, Centos_7.5.x_64, Ubuntu20.04_64 版本上验证测试, CompassStudio 运行正常。如遇安装问题, 请参考 [9 常见问题](#)。

2.3 CompassStudio 开发环境

使用 CompassStudio 之前, 需要配置其开发环境。CompassStudio 目前支持 Compass Zhouyi Z2/Z3/X1/X2 系列的 BuildTool, 以及 Zhouyi Z2/Z3/X1/X2 ToolChain。
CompassStudio 环境配置如下:

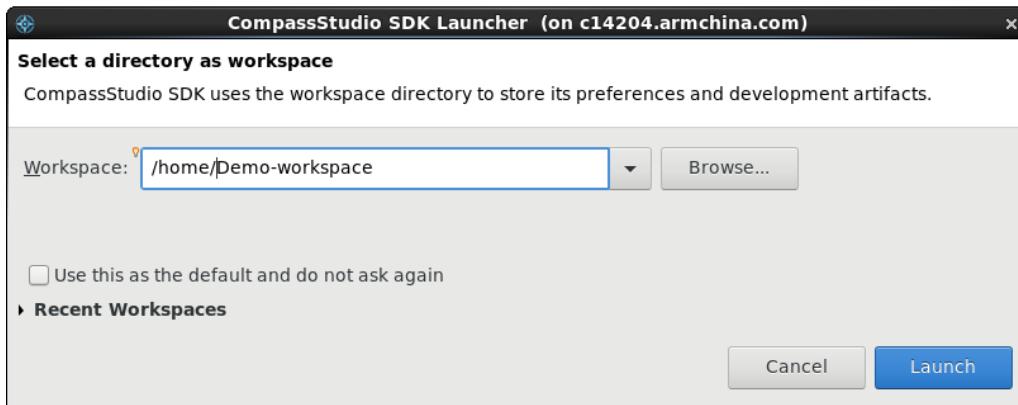
启动命令

进入解压后的 CompassStudio 文件夹下, 执行 “CompassStudio” 命令, 即可启动 CompassStudio。或者将 CompassStudio 设置成系统环境变量, 请参考 [2.2 CompassStudio 系统运行环境](#)。

配置 Workspace

第一次启动 CompassStudio 后, 将会弹出对话框要求设置 Workspace 目录路径。该目录将用于存放后续创建的项目工程文件, 如图 2-2 所示。

图 2-2: CompassStudio Workspace



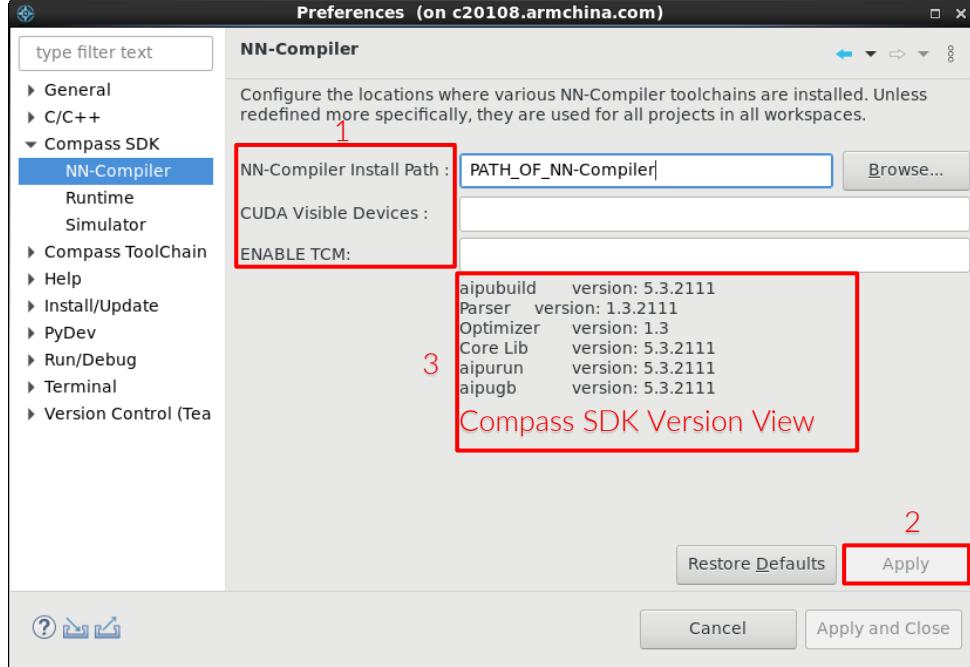
设置好 Workspace 目录之后, 单击 Launch, 将会启动 CompassStudio。

配置 Compass SDK 环境

1. 在菜单栏单击 Window > Preferences。
2. 如图 2-3 所示, 选择 Compass SDK > NN-Compiler。
3. 依次配置 NN-Compiler 的安装目录, GPU 安装槽位 (可选), TCM 环境 (可选)。
4. 配置完信息后, 单击 Apply。

“Compass SDK Version View” 中会显示 NN-Compiler 的版本。

图 2-3: 配置 NN-Compiler 环境



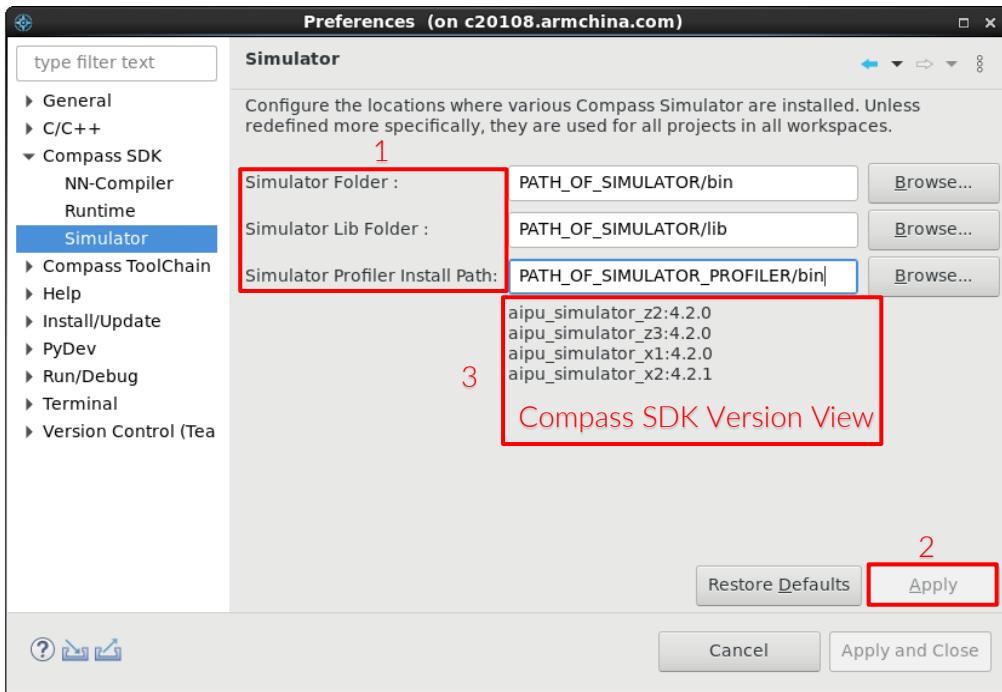
- 请留意不要混淆 NN-Compiler 和 C-Compiler。
- 如果 NN-Compiler 没有安装在默认的位置 (例如 `pip install` 使用了 `--prefix <dir>` 或者 `--target <dir>` 选项)。请将对应的安装路径添加到环境变量 `PYTHONPATH` 中。

配置 Simulator 环境

1. 在菜单栏单击 Window > Preferences。
2. 如图 2-4 所示, 选择 Compass SDK > Simulator。
3. 依次配置 Simulator 和 Simulator profiler 安装地址, 以及 Simulator 依赖的 lib 地址。
4. 配置完信息后, 单击 Apply。

“Compass SDK Version View” 中会显示 Simulator 的版本。

图 2-4：配置 Simulator 环境

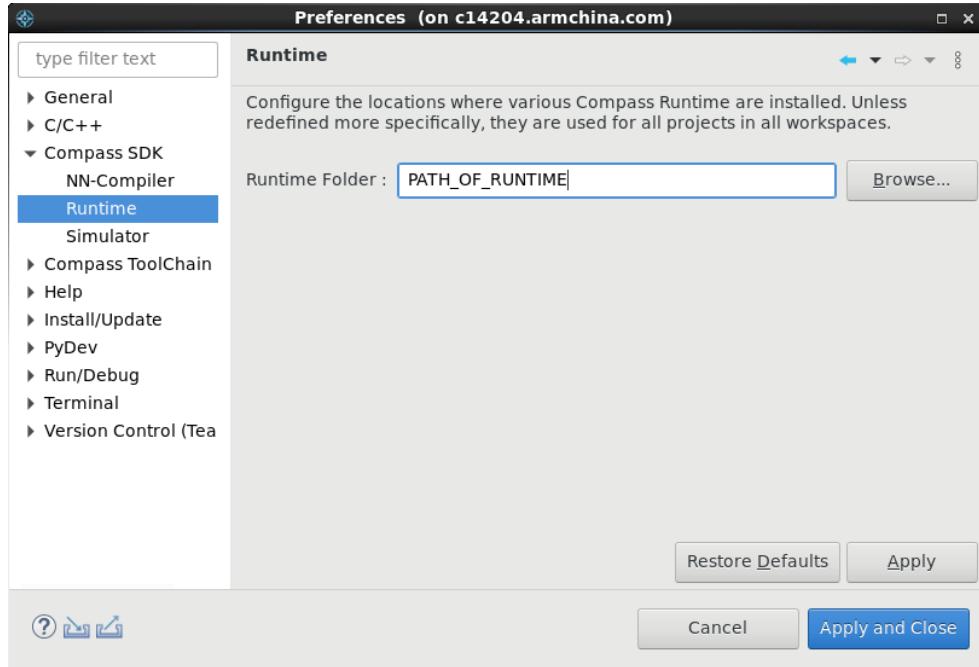


配置 Runtime 环境

周易 X2 ToolChain 在运行 Debug 时，需要加载 Runtime 的环境。如果不使用周易 X2 ToolChain 的相关功能，可以不配置该环境。具体操作如下：

1. 在菜单栏单击 Window > Preferences。
2. 如图 2-5 所示，选择 Compass SDK > Runtime。
3. 配置 Runtime 安装地址。
4. 配置完信息后，单击 Apply 即可。

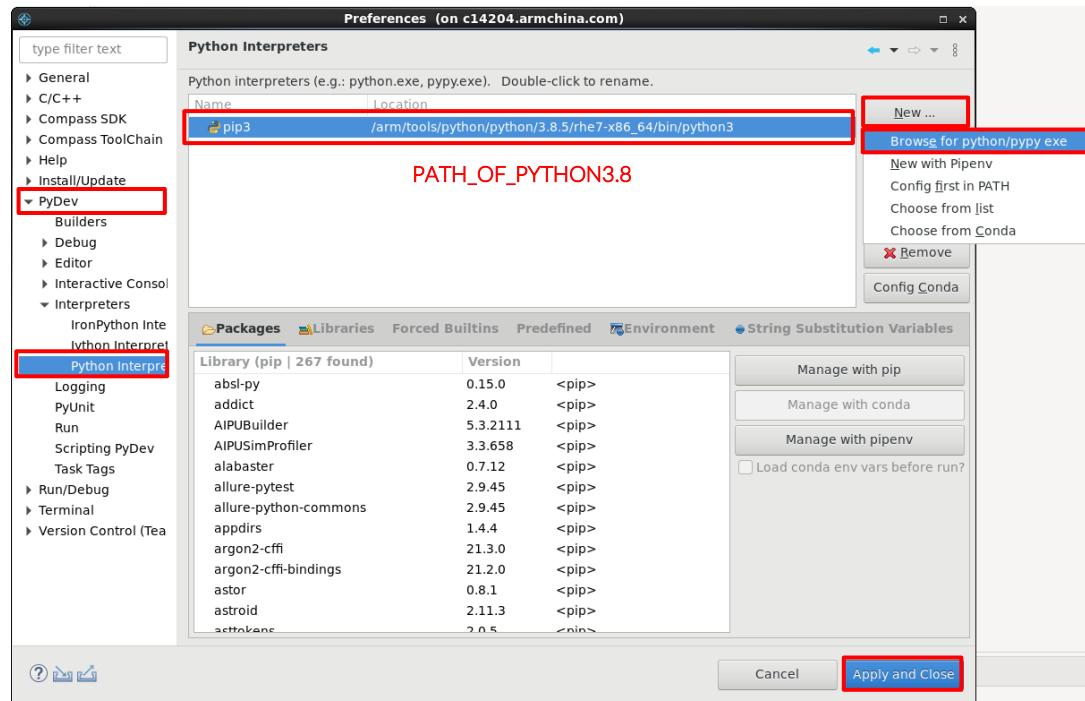
图 2-5: 配置 Runtime 环境



配置 Python 环境

- 如图 2-6 所示, 选择 PyDev > Interpreters > Python Interpreters 配置 Python 路径。

图 2-6: 配置 Python 环境



2. 单击 **New > Browse for python/pypy.exe**, 选择 Python 地址。
注意 Python 版本为 3.8.x。
3. 配置完 Python 相关环境后, 单击 **Apply and Close** 即可保存配置。

配置 ToolChain 环境

1. 配置 ToolChain 安装地址。

- 使用周易 Z2/Z3/X1 ToolChain, 需要配置 ToolChian 的环境变量。
CompassStudio 会自动加载该环境。

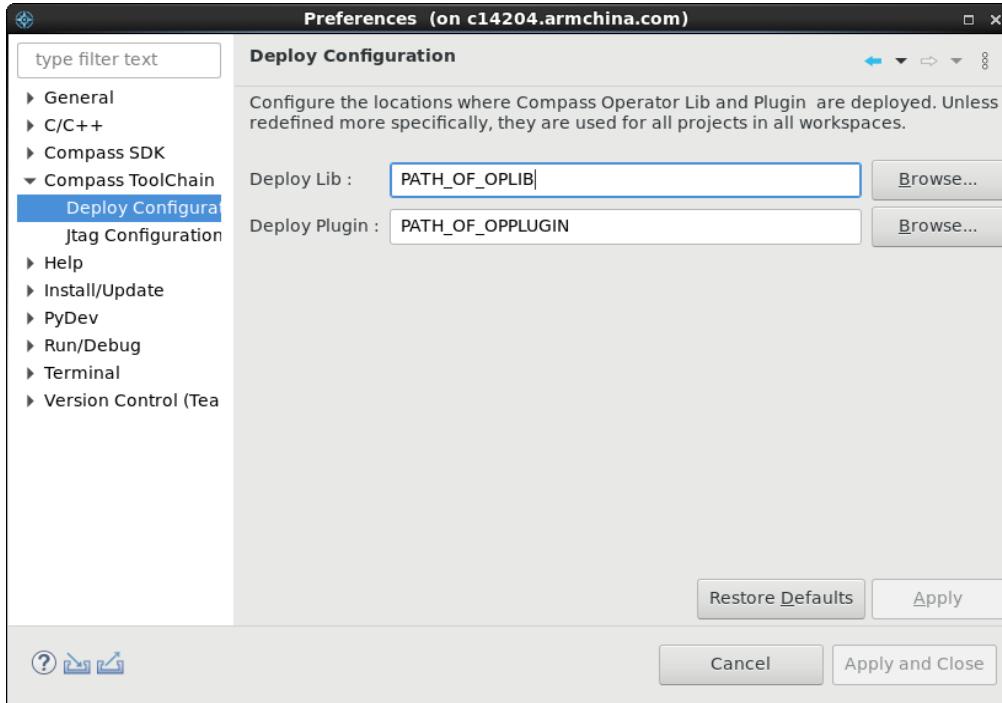
```
setenv PATH COMPILER_INSTALL_PATH/bin:DEBUG_INSTALL_PATH/bin:$PATH  
setenv LD_LIBRARY_PATH COMPILER_INSTALL_PATH/lib: DEBUG_INSTALL_PATH/lib:  
$LD_LIBRARY_PATH
```

- 使用周易 X2 ToolChain, 需要配置 ToolChian 的环境变量。CompassStudio 会自动加载该环境。

```
setenv PATH X2_COMPILER_INSTALL_PATH/bin:X2_DEBUG_INSTALL_PATH/bin:  
$PATH  
setenv LD_LIBRARY_PATH X2_COMPILER_INSTALL_PATH/lib: X2_DEBUG_INSTA  
LL_PATH/lib:  
$LD_LIBRARY_PATH
```

2. 配置算子部署的环境。

单击 **Window > Preferences > Compass ToolChain > Deploy Configuration** 进入配置算子部署路径的界面, 如图 2-7 所示。

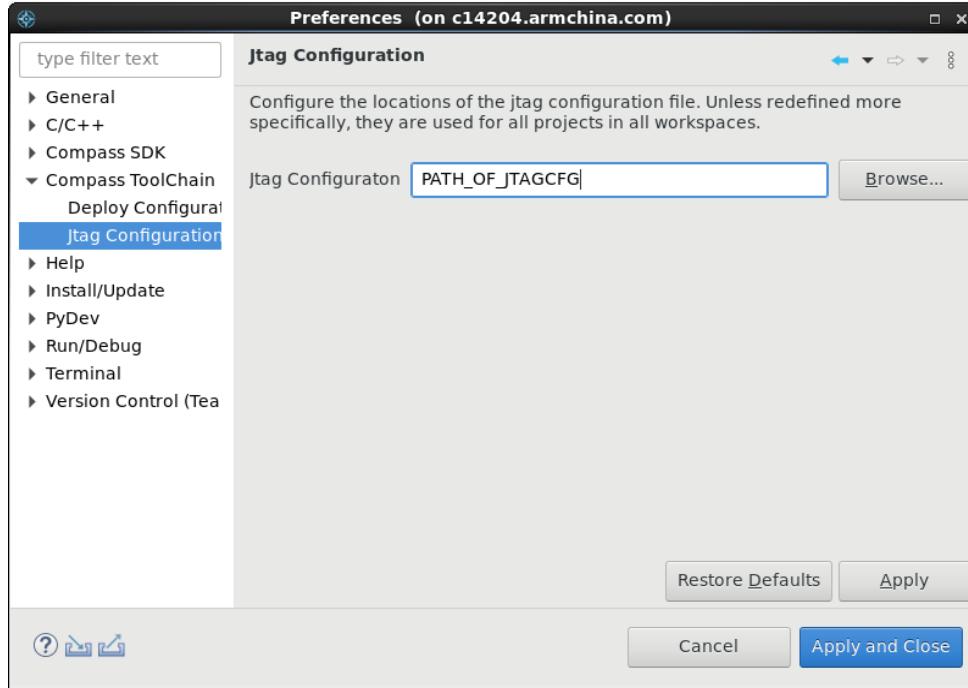
图 2-7: 配置算子部署的环境

Deploy Lib: 算子 (.out/.o) 部署的地址。

Deploy Plugin: 算子 plugin 部署的地址。

3. 配置 Jtag 文件地址。

单击 Window > Preferences > Compass ToolChain > Jtag Configuration 进入 Jtag 配置界面, 如图 2-8 所示。

图 2-8: 设置 Jtag 配置文件的地址

Jtag 配置文件默认存放在 Compass Debug Release 包的 config 目录下，您也可以自定义 Jtag 配置文件的地址。

3 模型工程的使用

本章节主要介绍 CompassStudio 的工程创建方式，以及 NN-Model 配置参数设置。

3.1 创建工程

在 CompassStudio 中创建项目有以下几种常见方式：

- 使用模板自动创建项目

这是最简单快捷的方式。CompassStudio 已经预先设置了模板项目。

- 从已有项目直接导入创建新项目

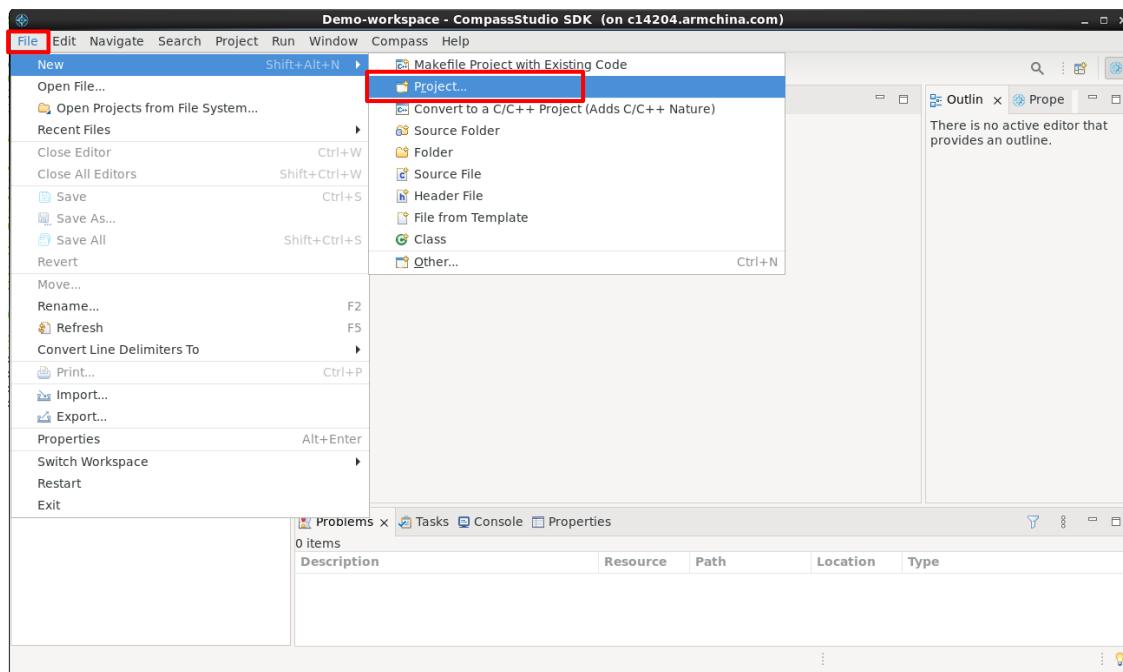
用户 A 可以将 CompassStudio 导出的项目文件夹直接进行打包保存，然后分享传播，用户 B 可以在其他电脑上直接导入该项目，从而以此为基础创建新的项目，并可以直接使用或者修改。

3.1.1 使用模板自动创建项目

使用模板自动创建项目的步骤如下：

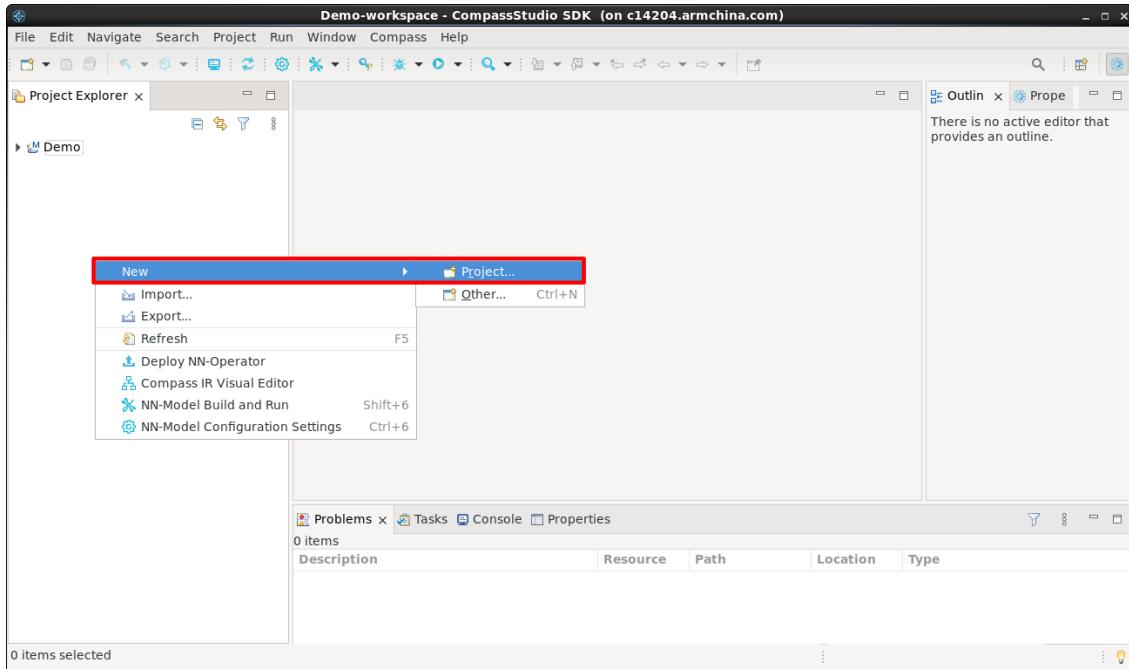
1. 在主界面上，单击 File > New > Project 创建新工程，如图 3-1 所示。

图 3-1：从菜单栏创建工程



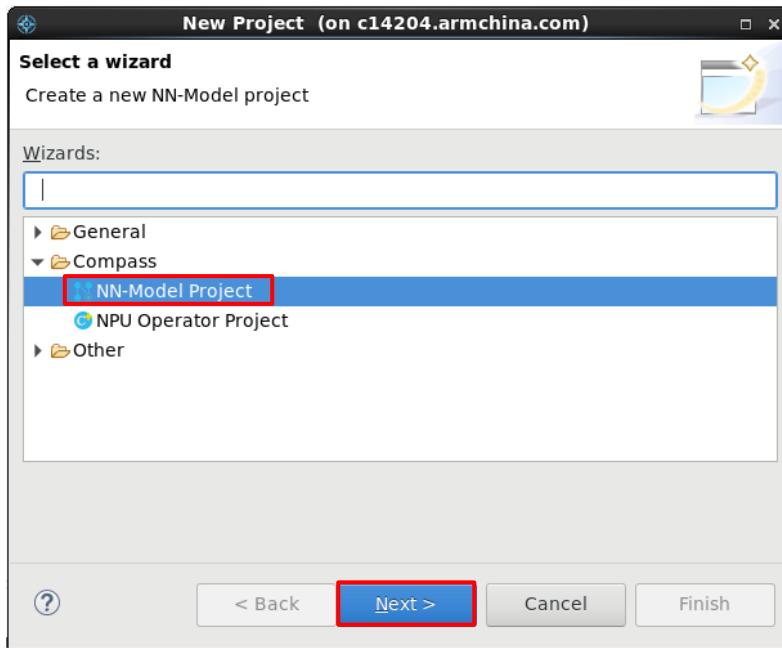
- 也可以右键单击左侧工程管理列表区，然后选择 New > Project，如图 3-2 所示。

图 3-2：使用鼠标右键创建工程

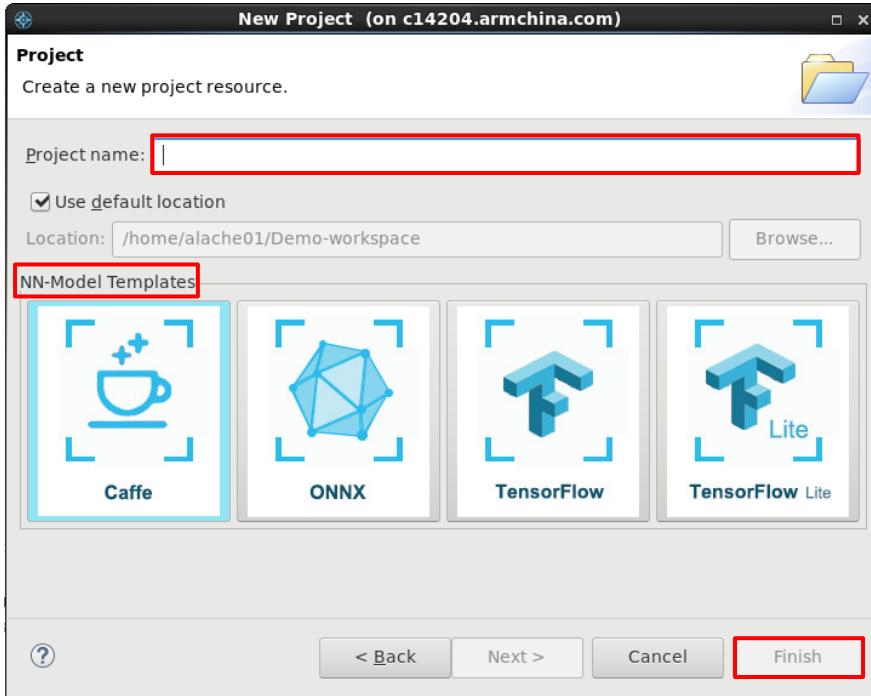


2. 在弹出的窗口中选择项目类型，如图 3-3 所示。

图 3-3：选择项目类型

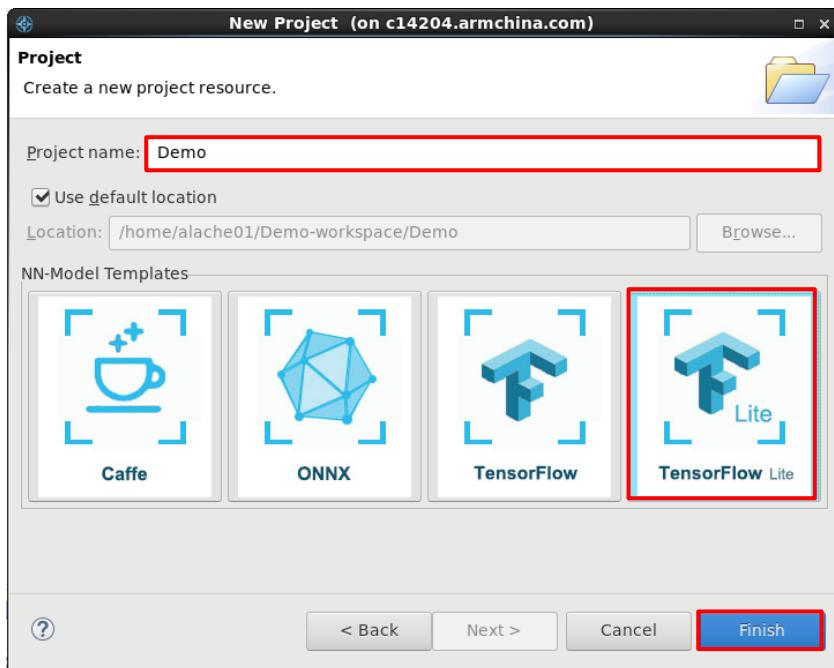


3. 单击 Next，在弹出的界面中选择模板工程，如图 3-4 所示。

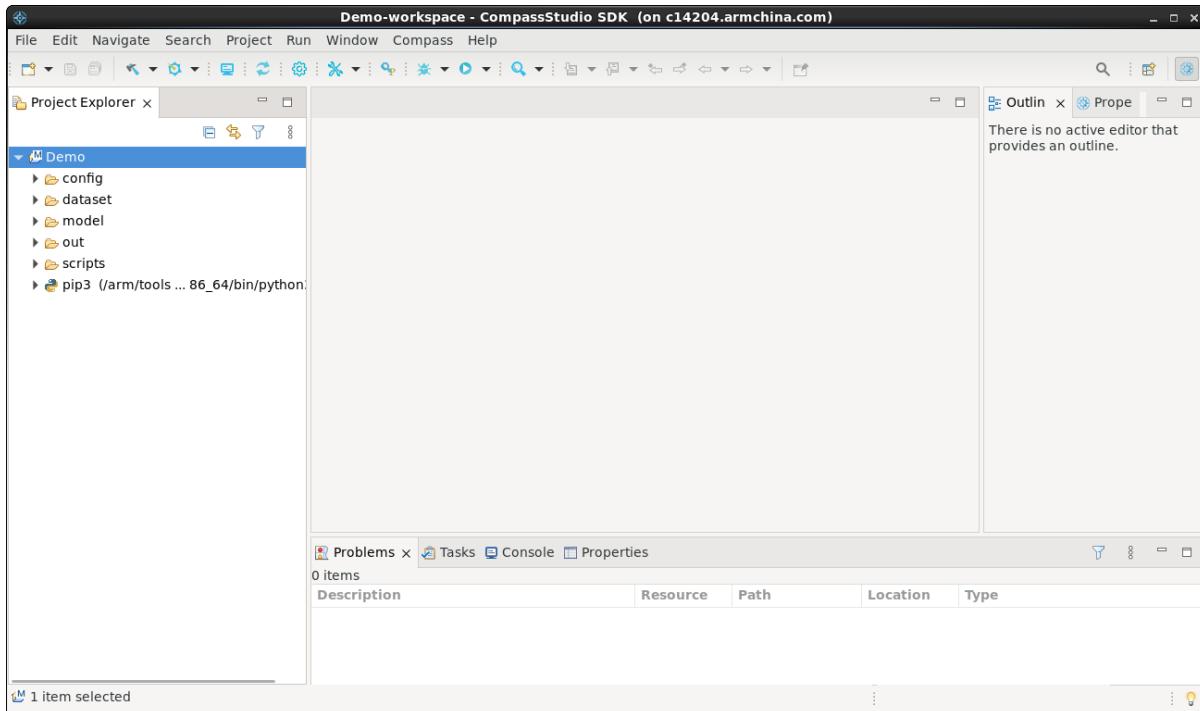
图 3-4: 选择项目类型**表 3-1: 新工程项目参数**

参数	描述
Project name	自定义项目名。
Use default location	如果勾选了此选项，则会使用默认 Workspace 文件夹存放项目。
NN-Model Templates	CompassStudio 模板项目。

4. 选择模板工程（以 TFLite 为例），然后单击 **Finish**，如图 3-5 所示。

图 3-5: 选择模板工程

至此，使用模板自动创建项目已经完成，如图 3-6 所示。

图 3-6: CompassStudio IDE 工程视图

工程结构

项目中包含 config、dataset、model、scripts、out 目录。

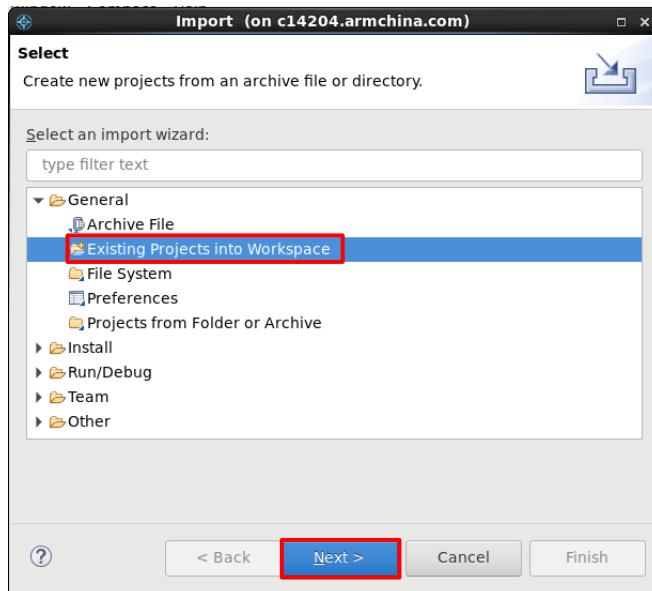
表 3-2: 工程目录结构

目录	描述
config	文件夹下包含 NN-Model Run, Build, Parse, Opt 所需要的 config 文件, 分别对应 run.cfg, build.cfg, parser.cfg, opt.cfg。
dataset	此文件夹包含推理的数据集和校准 label 文件。
model	此文件夹包含 model 文件。
scripts	此文件夹包含前处理脚本 (preprocess_dataset.py), 后处理脚本 (print_result.py) 和 run on hardware (run.sh) 的脚本。run.sh 包含 runtime 路径, aipu driver 路径, LD_LIBRARY_PATH 路径。
out	NN-Model Build 和 Run 的输出结果。

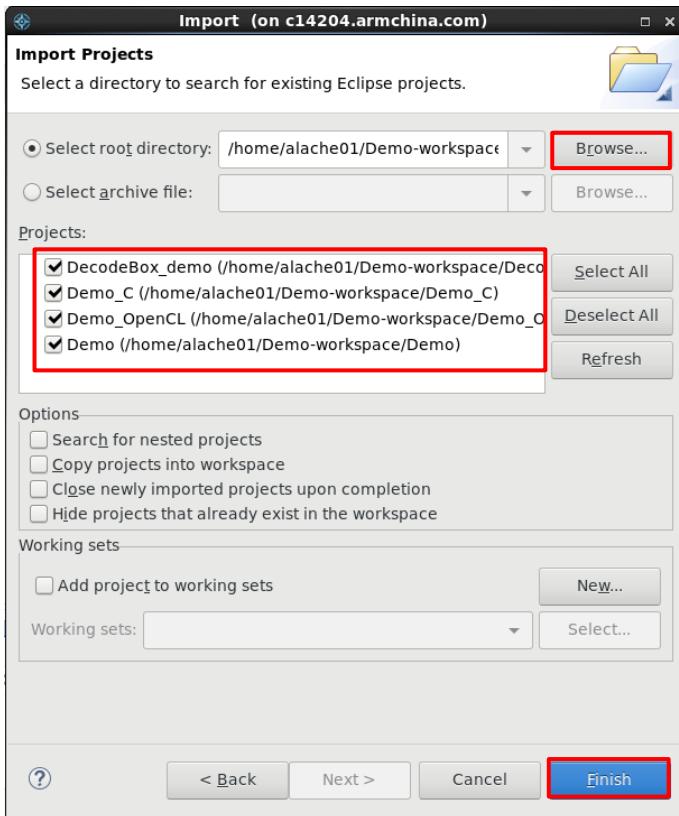
3.1.2 从已有项目直接导入创建项目

- 右键单击 CompassStudio 左侧的工程管理列表区, 选择 Import。在弹出的界面中选择 General > Existing Projects into Workspace, 如图 3-7 所示。

图 3-7: 导入工程



- 单击 Next, 进入下一步界面。
- 单击 Browse, 选择需要导入项目的地址, 在 Projects 区域选择需要导入的项目。然后单击 Finish, 完成项目的导入, 如图 3-8 所示。

图 3-8: 选择项目地址

导入的工程只能是 CompassStudio 导出的工程。非 CompassStudio 导出的工程无法正常导入。

3.2 NN-Model 配置参数设置

本章节主要介绍 cfg 文件的可视化及增删改查等功能。

配置 cfg 文件有两种方式：

- 手动编辑 cfg 文本
- 通过 NN-Model 配置界面进行编辑

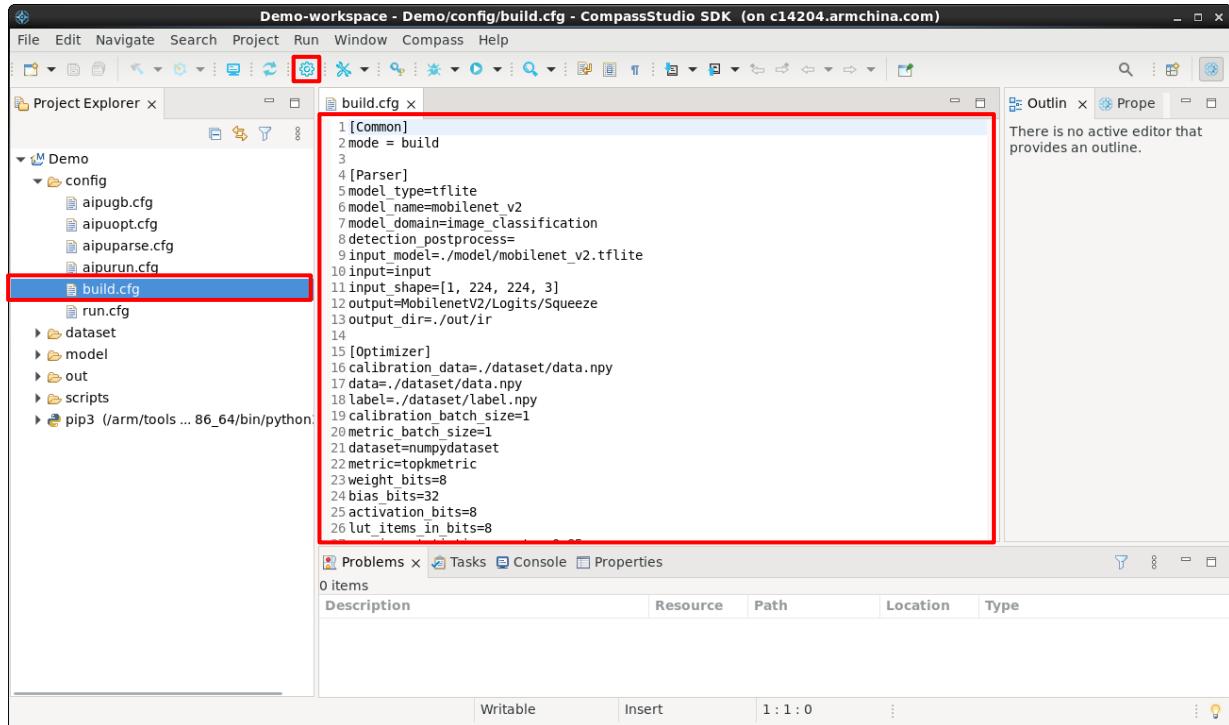
3.2.1 手动编辑 cfg 文本

1. 在 Project Explorer 列表中，打开想要编辑的 xxx.cfg 文件，在 cfg 文本内直接进行修改。

2. 修改完成后，选中工程，然后在工具栏单击 **NN-Model Configuration Settings** 图标，如图 3-9 所示。

弹出的界面中会将手动修改后的 cfg 进行同步展示。

图 3-9：手动编辑 cfg 文件

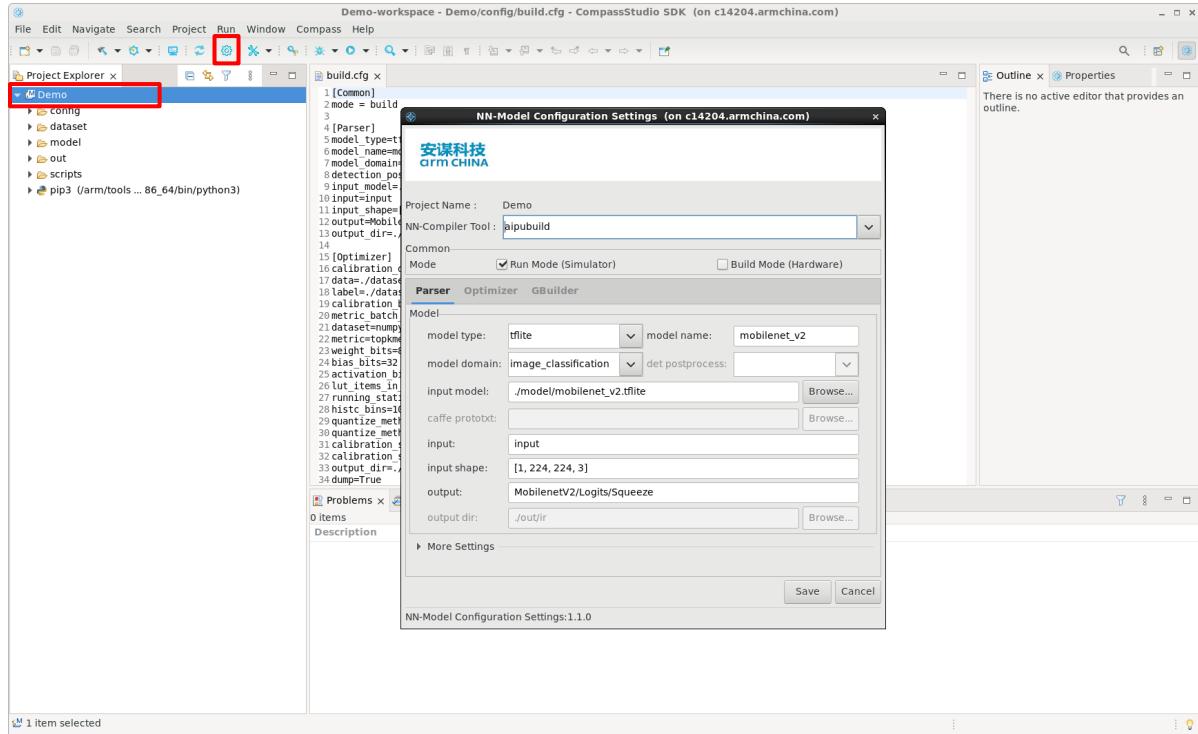


3.2.2 通过 NN-Model 配置界面进行编辑

您可以直接在 NN-Model 配置界面进行修改，修改后会同步到对应的 cfg 文件中。

1. 在 Project Explorer 列表中，选中想要编辑的工程，然后在工具栏单击 **NN-Model Configuration Settings** 图标（或右键单击想要编辑的工程，然后选择 **NN-Model Configuration Settings**），显示如下界面。

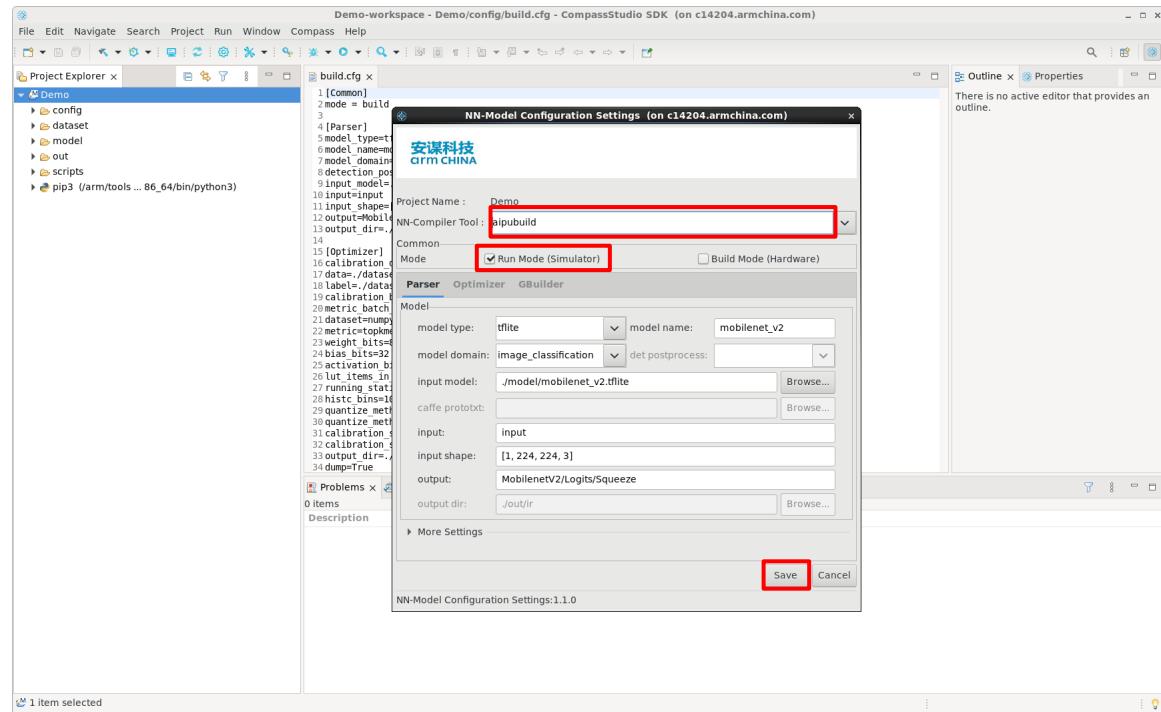
图 3-10: NN-Model 配置界面



2. 在 NN-Model 配置界面，编辑配置文件：

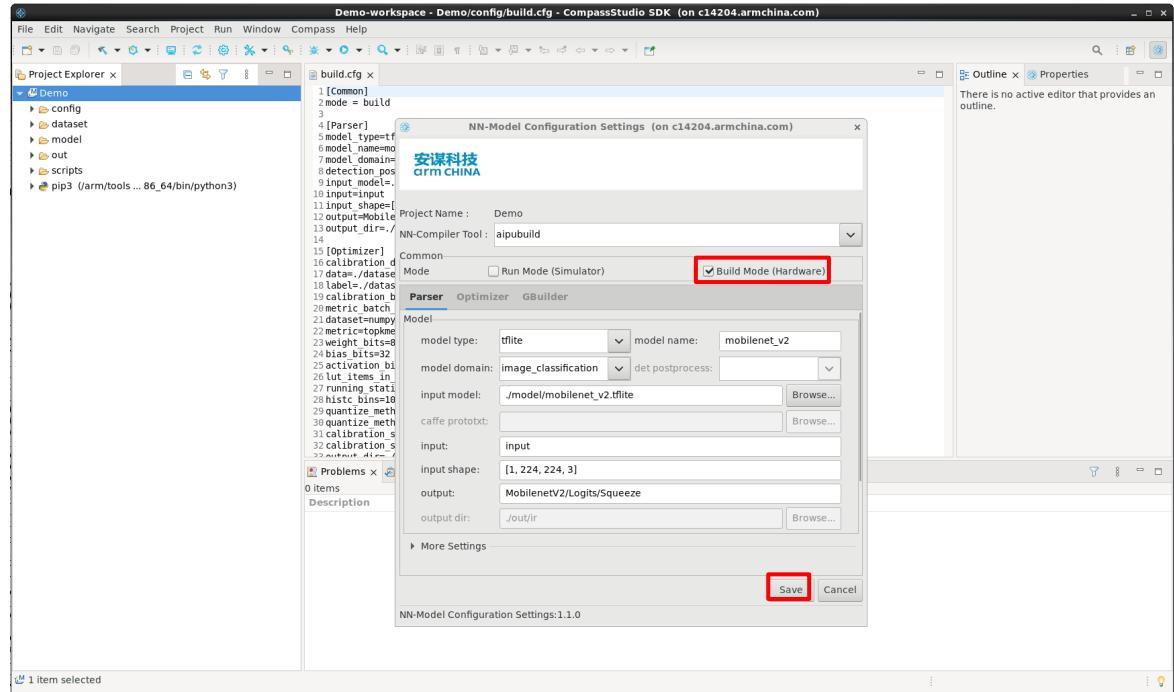
- Aipubuild Run cfg: 配置 NN-Model Run 需要的配置文件。
 - a. 在 NN-Compiler Tool 下拉列表中，选择 aipubuild。
 - b. 在 Mode 区域，选中 Run Mode (Simulator) 复选框。
 - c. 修改 Model 设置。

图 3-11: 修改 aipubuild Run 配置



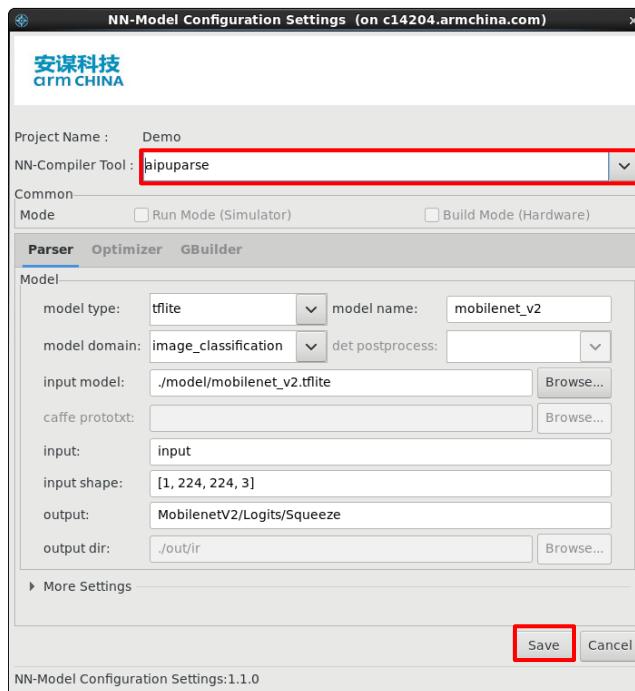
- Aipubuild Build cfg: 配置 NN-Model Build 需要的配置文件。
 - a. 在 NN-Compiler Tool 下拉列表中，选择 aipubuild。
 - b. 在 Mode 区域，选中 Build Mode (Hardware) 复选框。
 - c. 修改 Model 设置。

图 3-12: 修改 aipubuild Build 配置



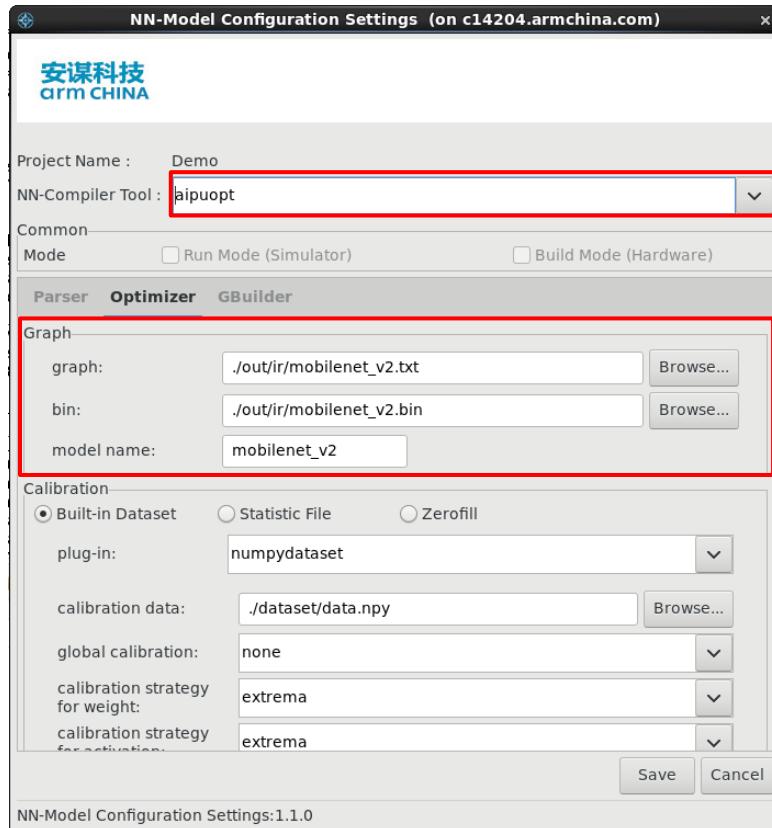
- Aipuparse cfg: 配置 NN-Model Parse 需要的配置文件。
 - 在 NN-Compiler Tool 下拉列表中, 选择 aipuparse。
 - 修改 Model 设置。

图 3-13: 修改 aipuparse 配置

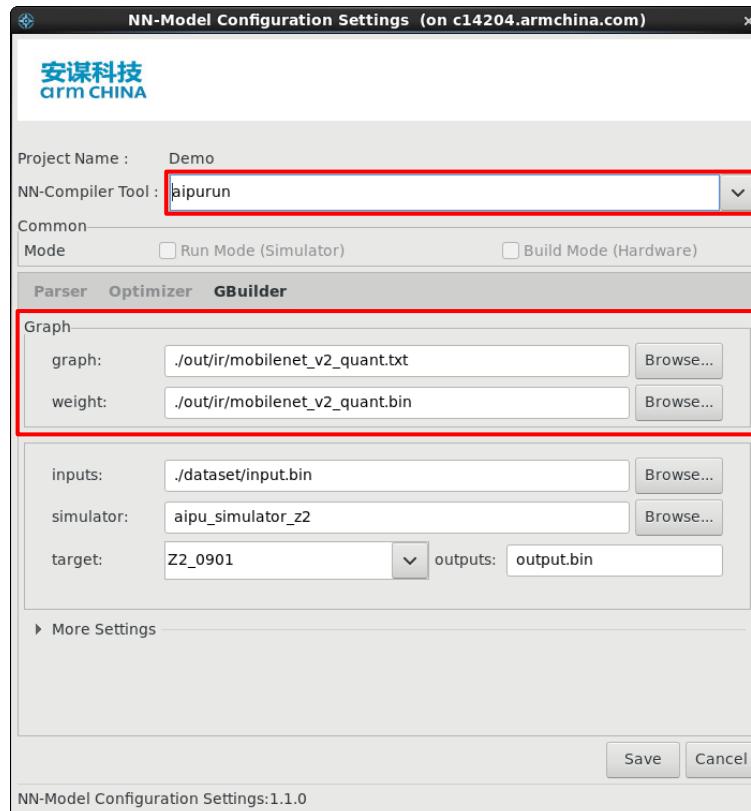


- Aipuopt cfg: 配置 NN-Model Opt 需要的配置文件。
 - 在 NN-Compiler Tool 下拉列表中, 选择 aipuopt。
 - 修改 Model 设置。

图 3-14: 修改 aipuopt 配置

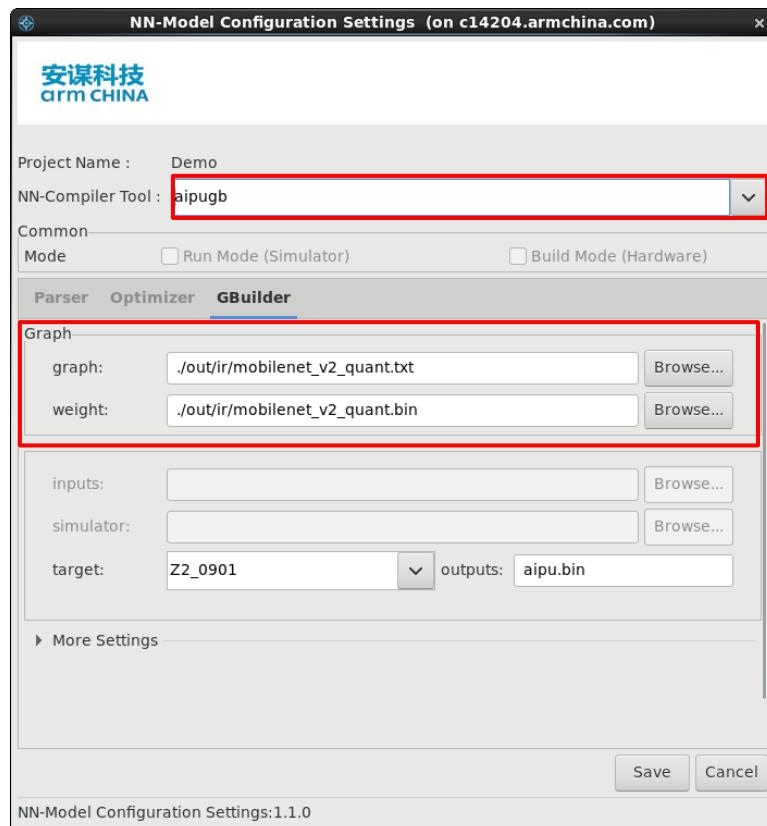


- Aipurun cfg: 配置 NN-Model IR Run 需要的配置文件。
 - 在 NN-Compiler Tool 下拉列表中, 选择 aipurun。
 - 修改 Model 设置。

图 3-15: 修改 aipurun 配置

- Aipugb cfg: 配置 NN-Model IR Build 需要的配置文件。
 - a. 在 NN-Compiler Tool 下拉列表中，选择 aipugb。
 - b. 修改 Model 设置。

图 3-16: 修改 aipugb 配置



3. 在配置页面完成配置后，单击 Save，即可更新 cfg 文件。

表 3-3: 参数描述

参数	描述
Project Name	选定的工程名称。
NN-Compiler Tool	此模块集成了 aipubuild, aipuparse 和 aipuopt。
Mode	Run Mode 是指 run on simulator, 对应 run.cfg。Build Mode 是指 run on hardware, 对应 build.cfg。选择对应的模式后，配置界面会加载对应的 cfg 文件，并展示对应的参数信息。在配置界面对 parser、optimizer、GBuilder 进行配置，配置完成并保存后，可以同步到对应模式的 cfg 文件中。 有关 parser、optimizer、GBuilder 各模块参数的使用规范，请参考《Arm China Zhouyi Compass NN Compiler User Guide》中的“Chapter 4 Using the NN Compiler”章节。

4 Model Build 和 Run

本章节主要介绍 NN-Model Run、NN-Model Build、Run on Simulator、Run on Hardware、Profiler 等基本功能，以及 Compass IR Build、Compass IR Run on Hardware、Compass IR Run、Compass IR Visual Editor、QuantTuning 等高级功能。

基本功能

- NN-Model Run: 将 Model 编译为可在 Simulator 运行的 bin 文件。NN-Model Run 会直接运行在 Simulator 上产生 output.bin。因此，NN-Model Run 包含了 Run on Simulator 的过程，其对应 Compass SDK 的 aipubuild (Simulator Mode) 工具。
- NN-Model Build: 将 Model 编译为可在 Hardware 上运行的 bin 文件。运行 NN-Model Build 并不会直接在 Hardware 上运行，只会产生 Build 后的 bin 文件，其过程对应 Compass SDK 的 aipubuild (Hardware Mode) 工具。
- Run on Hardware: 在 Hardware 上运行网络得到输出，将产生的 output.bin 文件进行解析推理，对比 dataset 的 label，推理出识别结果。
- Profiler: 获取在 Simulator/Hardware 上运行的整个网络的性能信息。

高级功能

- Compass IR Build: 将 Model 通过 Parser, OPT 阶段处理产生 Float IR 和 Quant(int8) IR，最后由 Compass IR Build 产生可以在 Hardware 上运行的 bin 文件，并不会直接运行在 Hardware 上，其过程对应 Compass SDK 的 aipuparse、aipuopt、aipugb 工具。
- Compass IR Run: 将 Model 通过 Parser, OPT 阶段处理产生 Float IR 和 Quant(int8) IR，最后由 Compass IR Run 直接运行在 Simulator 上产生 output.bin，其过程对应 Compass SDK 的 aipuparse、aipuopt、aipurun 工具。
- Compass IR Run on Hardware: 将 Compass IR Build 产生的 bin 文件放在 Hardware 上运行，其过程和 [4.3 Run on Hardware](#) 一致。
- Compass IR Run: 将 Model 通过 Parser, OPT 阶段处理产生 Float IR 和 Quant(int8) IR，最后由 Compass IR Run 直接运行在 Simulator 上产生 output.bin，其过程对应 Compass SDK 的 aipuparse、aipuopt、aipurun 工具。
- QuantTuning: 显示 OPT 阶段 quant 调优的信息以及整个网络相似度趋势。
- NpyFile Visualizer: 显示 Float 32 和 Quant npy 文件对比的趋势图。
- Compass IR Visual Editor: 显示网络 IR 的结构图，并且可以对每个网络层进行编辑。Float IR 的网络可以通过 Optimizer 进行精度调优配置。

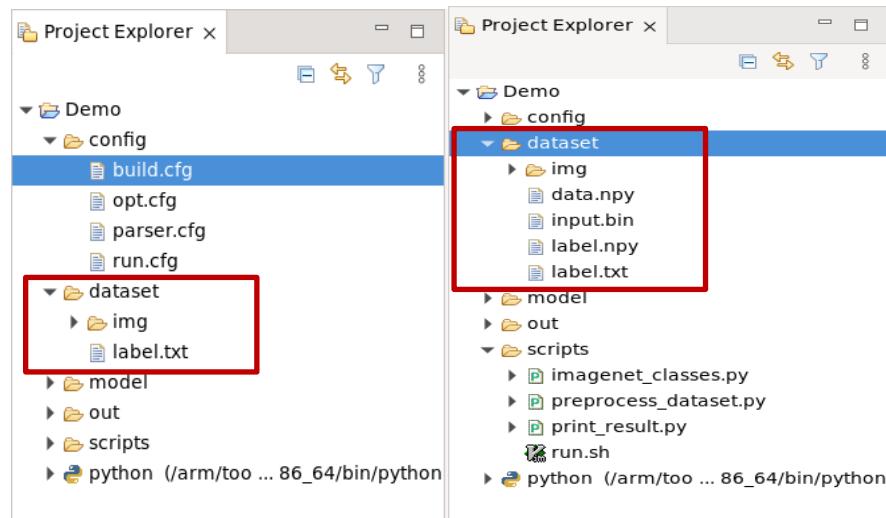
CompassStudio 运行 NN-Model，需要以下几个步骤：

1. 前处理
2. NN-Model Build 和 Run
3. Run on Simulator 和 Run on Hardware
4. 后处理

4.1 前处理

CompassStudio 提供 NN-Model 模板。完成工程创建后，针对 CompassStudio 提供的模板，您需要运行前处理脚本“scripts/preprocess_dataset.py”产生数据集。您可以右键单击该脚本，然后选择 **Run As > Python Run**，运行完前处理脚本后刷新工程，会在 dataset 文件夹下产生“data.npy”、“input.bin”、“label.npy”。dataset 文件夹前后对比如图 4-1 所示。

图 4-1：前处理后 dataset 对比



如果您使用自己的 Model 和 dataset，需要自定义前处理脚本。可以参考该前处理脚本，自行生成对应 Model 和 dataset 的前处理脚本。

4.2 NN-Model Build

CompassStudio 将 Model 编译成可以运行在 Hardware 上的 bin 文件，并不会直接运行在 Hardware 上。有关 Run on Hardware 的信息，请参考 4.3 章节。

1. 在 Project Explorer 列表中，选择工程，在工具栏单击 NN-Model Build and Run 图标（或右键单击工程，然后选择 NN-Model Build and Run），弹出界面如图 4-2 所示。

图 4-2: NN-Model Build 界面

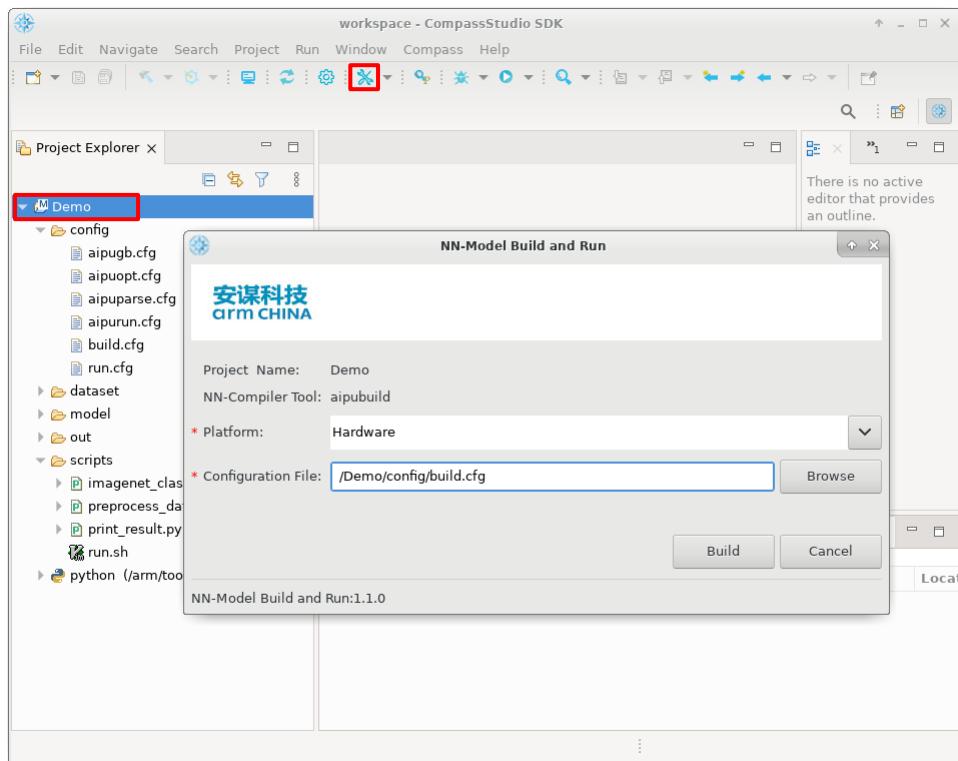
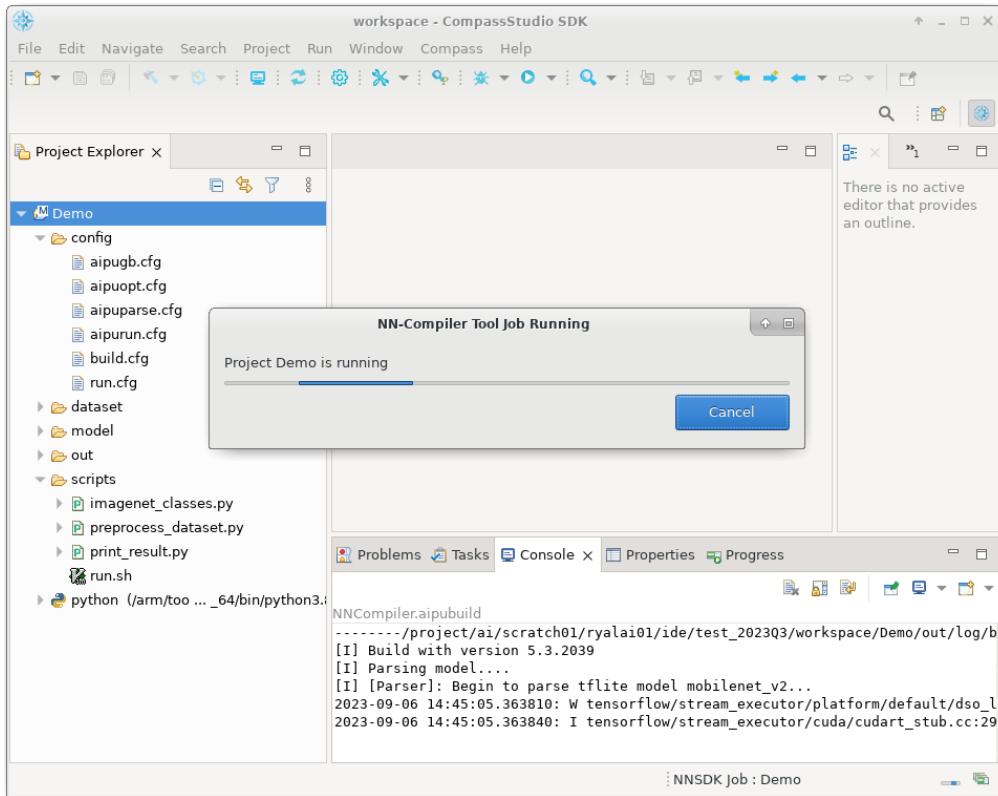


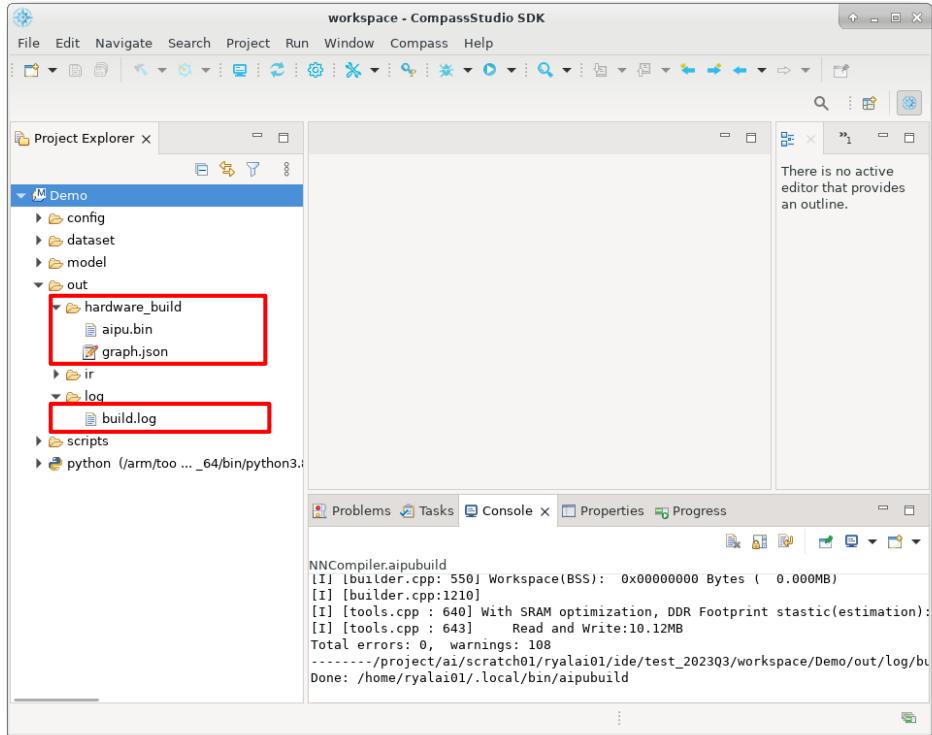
表 4-1: 参数描述

参数	描述
Project Name	选定的工程名称。
NN-Compiler Tool	NN-Compiler 工具，该模式下默认是 aipubuild。
Platform	选择 Build 在 Simulator 或 Hardware 上运行的 bin 文件。NN-Model Run 则选择 Simulator。
Configuration File	CompassStudio 会根据您选择的 Build 模式，自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件，单击 Browse，然后选择自定义的 cfg 文件路径。

2. 配置完 NN-Model Build 参数后，单击 Build，运行 NN-Model Build。Console 页签会实时输出日志，并且会弹出运行的进度条，如图 4-3 所示。

图 4-3: NN-Model Build 运行中

3. 运行结束后, 会弹出“Success”。您也可以在 Console 页签查看日志, 日志最后一行会显示运行 aipubuild build 的错误信息 (0 表示运行成功)。同时日志也会保存到工程的 out/log/build.log 文件, 如果“build.log”文件已经存在, 则会覆盖“build.log”产生新的日志信息。如果运行成功, 也会在工程的 out 下产生对应的 hardware_build 文件夹, 文件夹下有对应的“aipu.bin”文件, 如图 4-4 所示。

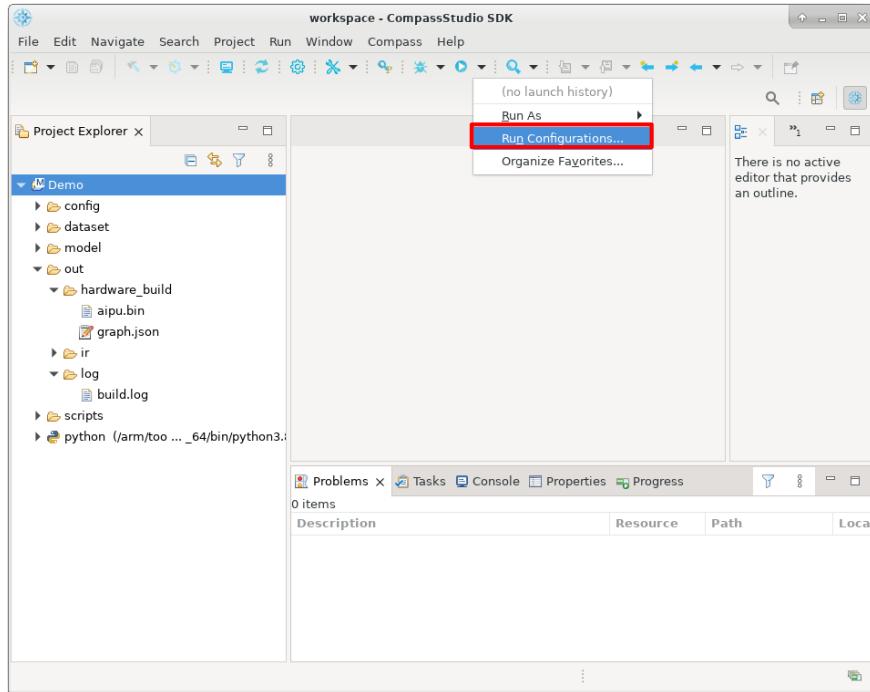
图 4-4: NN-Model Build 运行结果

4.3 Run on Hardware

4.3.1 Run on Hardware 环境配置

在硬件上运行 Compass SDK，需要先配置 Hardware 的相关信息，具体步骤如下：

1. 准备开发板环境，包括烧录，安装 Runtime 等。有关安装 Runtime 的详细信息，请参考《Arm China Zhouyi Compass Driver and Runtime User Guide》。
2. 确认 CompassStudio 跟开发板的连接方式（开发板支持 SFTP 传输协议）。默认是网线连接，下文操作基于网线连接。
3. 参考 [4.2 NN-Model Build](#) 或者 [Compass IR Build](#)，生成运行在 Hardware 上的 bin 文件。该文件位于 out/hardware_build 文件夹下。
4. 在 CompassStudio 配置开发板相关信息，具体操作如下：
 - a. 工具栏单击 Run Configurations，如图 4-5 所示。

图 4-5: Open Run on Hardware GUI

- b. 在如图 4-6 所示的界面，新建一个 NPU Hardware Run Application。

双击 NPU Hardware Run 或右键单击 NPU Hardware Run，然后选择 New Configuration，即可创建 Application。

图 4-6: Run on Hardware 信息配置页面

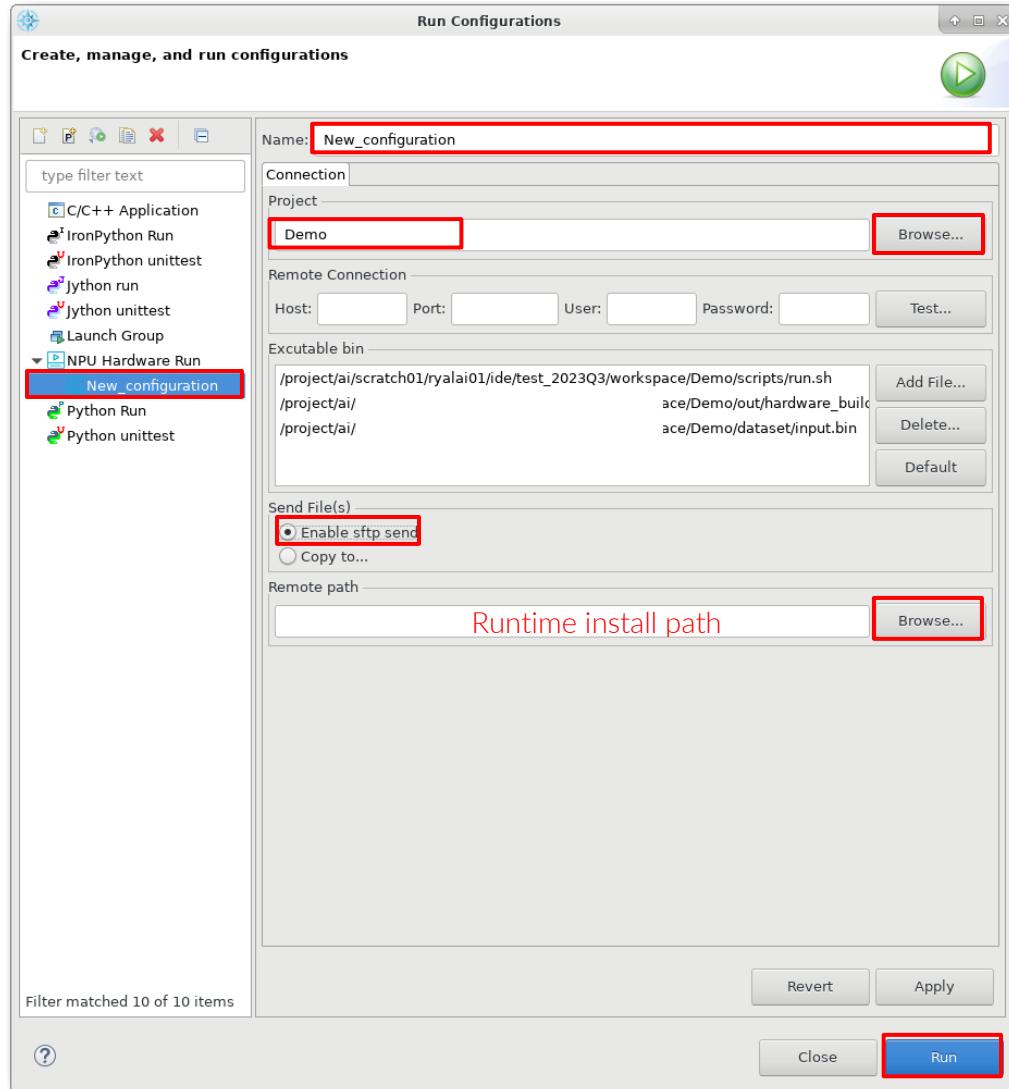


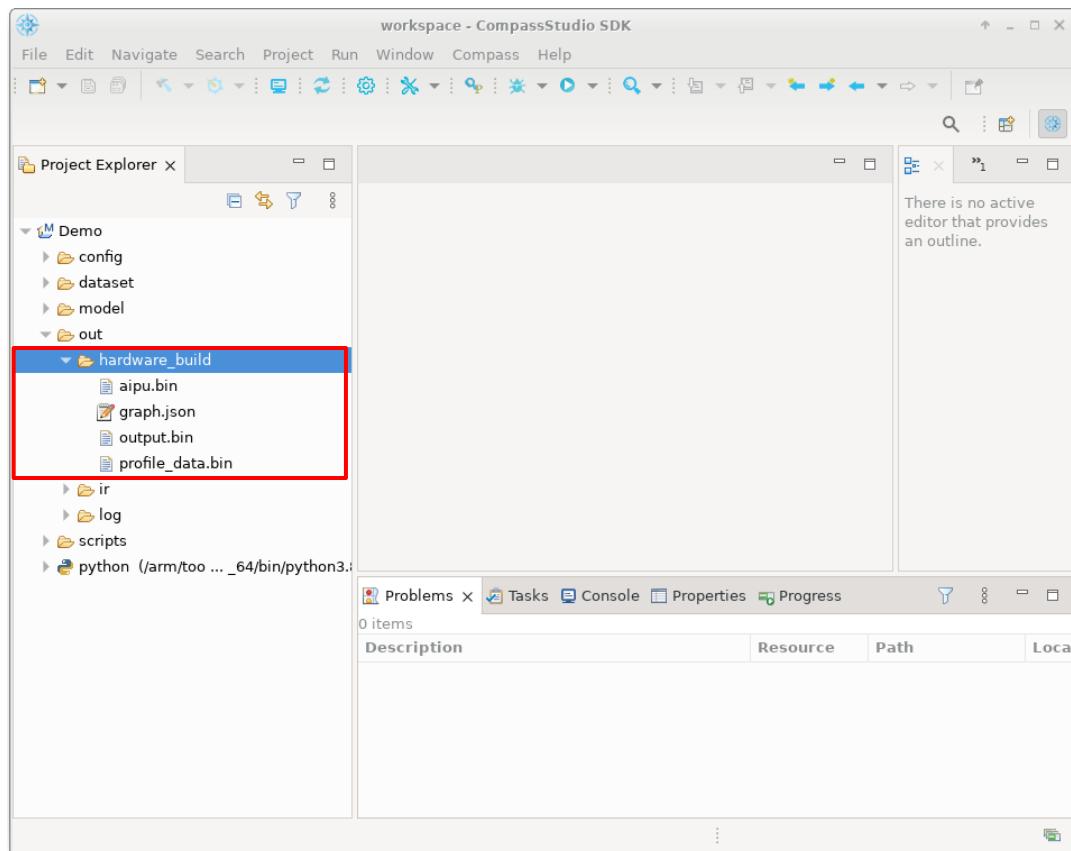
表 4-2: 配置参数描述

参数	描述
Name	自定义 Run on Hardware 应用程序名称。
Project	单击 Browse , 然后选择工程项目。选择完工程项目, CompassStudio 会自动将运行在 FPGA 上的脚本、build 产生的 Hardware bin 文件和 Input bin 文件填充到“Executable bin”模块中。
Remote Connection	配置开发板的通信信息。配置完成后, 单击 Test , 可以测试是否连接到开发板。
Executable bin	开发板上运行 Compass SDK 的数据文件及脚本。可以单击 Add File 自定义增加数据或者脚本。
Remote path	开发板上 Runtime 安装的地址。单击 Browse , 选择开发板 Runtime 安装地址即可。这里的远程路径要跟 run.sh 里配置的 Runtime 路径和 lib 路径一致。

参数	描述
Send Files	<p>可以选择通过网线或者 SD 卡拷贝的方式。</p> <ul style="list-style-type: none"> 如果使用网线, 选中 Enable sftp send, CompassStudio 自动将 Executable bin 中的数据拷贝到开发板对应配置的 Remote path 的目录中, 单击 Apply > Run, 将会在开发板运行 buildtool。 如果没有使用网线连接开发板, 可以使用 SD 卡拷贝, 单击 Copy to ..., CompassStudio 自动将 Executable bin 中的数据拷贝到 SD 卡。拷贝完成后需要手动将数据拷贝到开发板对应配置的 Remote path 目录中。此时, CompassStudio 因为没有和开发板互连, 并不能执行 buildtool 的操作, 需要去开发板执行拷贝过去的 run.sh, 最后需要将结果通过 SD 卡拷贝回 CompassStudio。

- c. 单击 **Apply** 保存配置, 然后单击 **Run**, 在 Hardware 上运行 Compass SDK。日志会实时打印在控制台。运行结束后, 将 Hardware 上的结果同步到 CompassStudio 的out/hardware_build 文件夹下, 如图 4-7 所示。

图 4-7: Run on Hardware 输出



至此, 在开发板上运行完成后, 产生 “output.bin” 同步到 CompassStudio 中。

4.4 Run on Simulator

Run on Simulator 主要包括两个步骤:

1. 编译产生在 Simulator 上运行的 bin 文件。
2. 将该 bin 文件运行在 Simulator 上产生输出。

Compass SDK 中的 NN-Model Run 则是集成了这两个步骤, 直接将 Model 编译后运行在 Simulator 上。

本小节主要介绍 CompassStudio 如何使用 NN-Model Run 执行 Run on Simulator。步骤如下:

1. 在 Project Explorer 列表中, 选择工程, 然后在工具栏单击 NN-Model Build and Run 图标 (或右键单击工程, 然后选择 NN-Model Build and Run), 弹出界面如图 4-8 所示。

图 4-8: NN-Model Run 界面

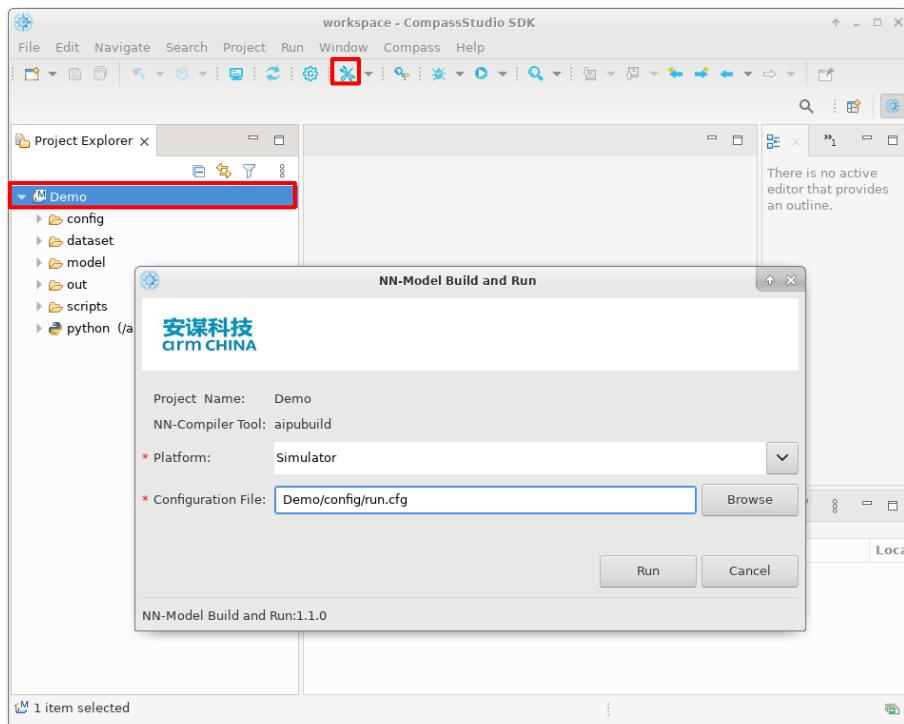


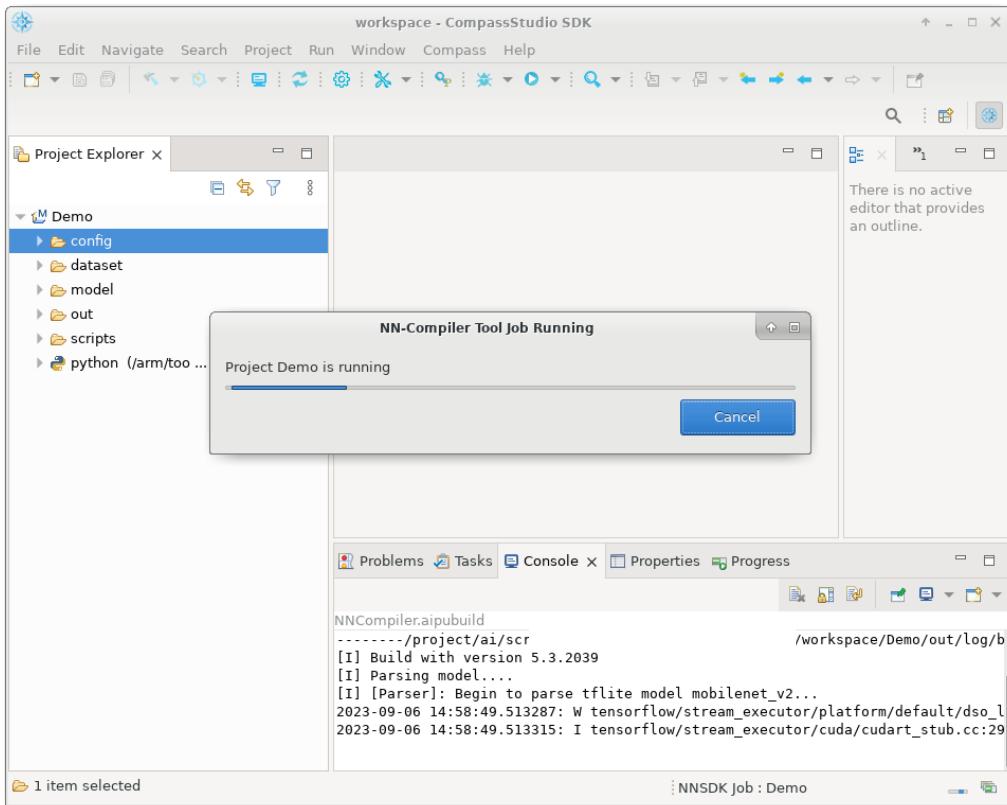
表 4-3: 参数描述

参数	描述
Project Name	选定的工程名称。

参数	描述
NN-Compiler Tool	NN-Compiler 工具, 该模式下默认是 aipubuild。
Platform	选择 Build 在 Simulator 或 Hardware 上运行的 bin 文件。NN-Model Run 则选择 Simulator。
Configuration File	CompassStudio 会根据您选择的 Build 模式, 自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件, 单击 Browse, 然后选择自定义的 cfg 文件路径。

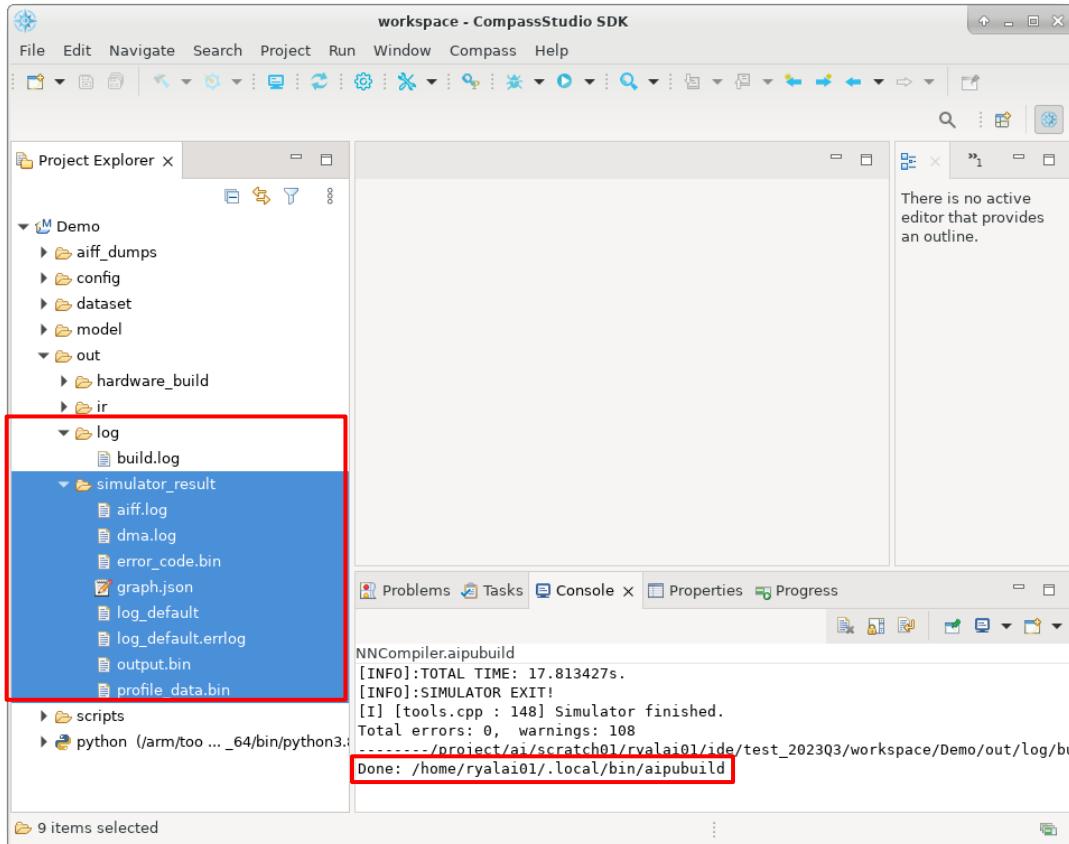
2. 配置完 NN-Model Run 参数后, 单击 Run, 运行 NN-Model Run。Console 页签会实时输出运行日志, 并且会弹出运行的进度条, 如图 4-9 所示。

图 4-9: NN-Model 运行中



3. 运行结束后, 会弹出“Success”。您也可以在 Console 页签查看运行日志。日志最后一行会显示运行 aipubuild 的错误信息 (0 表示运行成功)。同时日志也会保存到工程的 out/log 文件夹下。如果运行成功, 也会在工程的 out 下产生对应的 simulator_result 文件夹, 文件夹下有对应的 output 文件 (如 bin、profiler data、png 等), 如图 4-10 所示。

图 4-10: NN-Model 运行结果



4.5 后处理

执行 4.3 或者 4.4 节的步骤后，会得到在 Simulator 和 Hardware 上的 output.bin。对运行结果进行推理，以 Simulator 的结果为例，具体步骤如下：

1. 后处理

- 使用 CompassStudio 的模板，主要进行以下配置：

打开“scripts”文件，修改“print_result.py”，设置为：

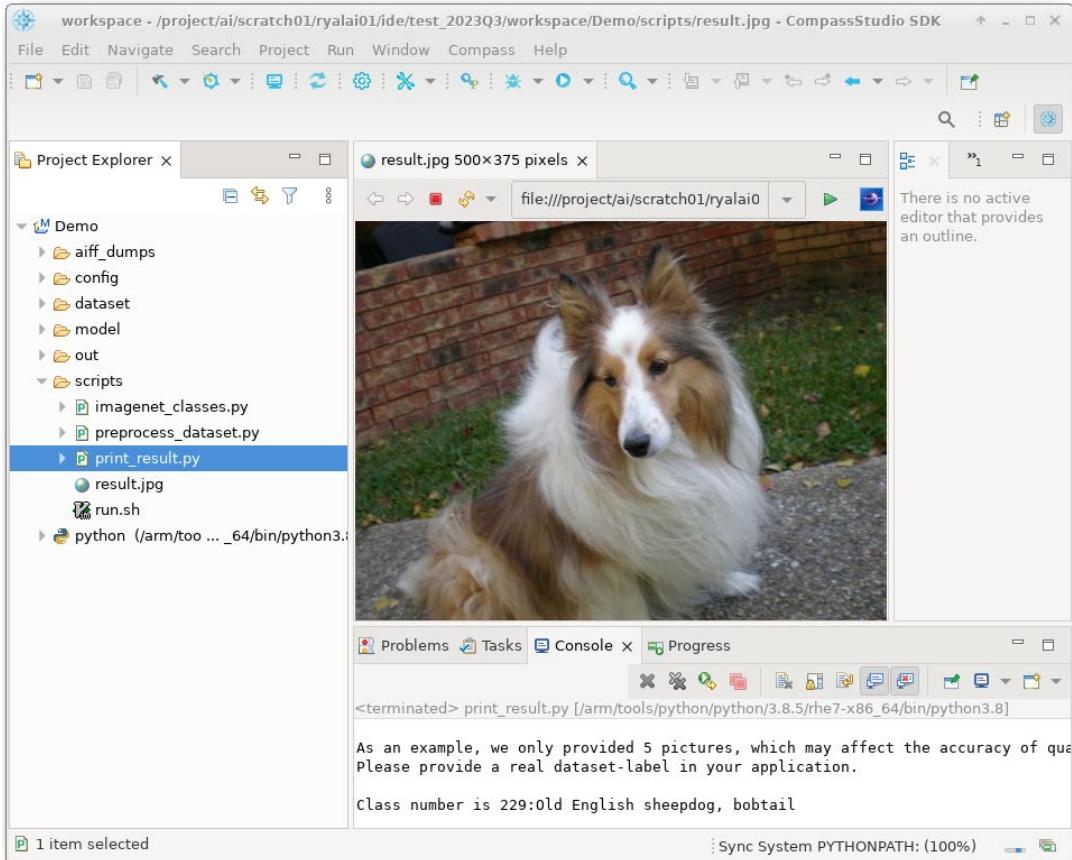
```
.....
.....
labelfile = '../out/simulator_result/output.bin' (output.bin 地址)
.....
.....
```

- 如果您使用自己的 Model 和 dataset，需要自行定义后处理脚本。请参考“scripts/print_result.py”。

2. 结果显示

右键单击“print_result.py”，然后选择 Run As > Python Run，结果如图 4-11 所示。

图 4-11: Run on Simulator 后处理结果



Hardware 后处理过程类似，只需要修改“print_result.py”，将“labelfile”指向 Hardware 上产生的“output.bin”文件。

4.6 Profiler

CompassStudio 提供 Profiler 功能，用来获取网络性能信息。目前 CompassStudio Profiler 模块包含 Simulator Profiler、Hardware Profiler、Multi-core profiler chart 等功能。

- Simulator Profiler 支持 Zhouyi Compass Z2、Z3、X1、以及 X2 系列。
- Hardware Profiler 支持 Zhouyi Compass Z2、Z3、X1、以及 X2 系列。
- Multi-core profiler chart 只支持 Zhouyi Compass X2_1204MP3。

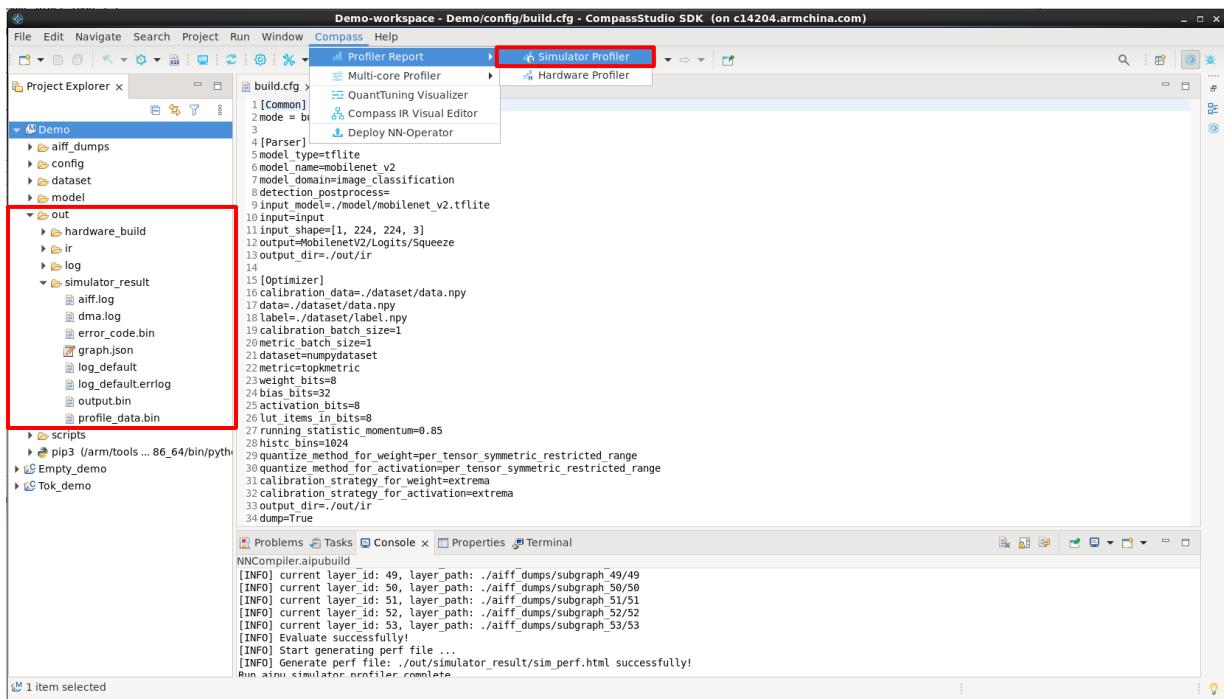
4.6.1 Simulator Profiler

针对 Run on Simulator 上的结果进行分析。若要解析“aipu.bin”里的 Profiler 功能，需要预先将“run.cfg”的配置文件里的 Profile 开关打开，生成的“aipu.bin”文件里将自动支持 Profiler 功能，同时也会自动生成一个“graph.json”文件。该文件记录了用来生成最终 Profiler 报告的节点信息。

在 CompassStudio 中执行 Run on Simulator (参考 4.4) 将会生成一个 buffer 文件 (“profiler_data.bin”)，此文件记录了运行时网络信息。根据这两个文件，CompassStudio 可自动生成一份可视化报告来展示网络各层的性能信息。

- 如图 4-12 所示，在 out/simulator_result 目录下已经生成了“aipu.bin”、“graph.json”、“profiler_data.bin”。在主菜单栏单击 **Compass > Profiler Report > Simulator Profiler** 将会生成 Simulator Profiler 可视化报告。

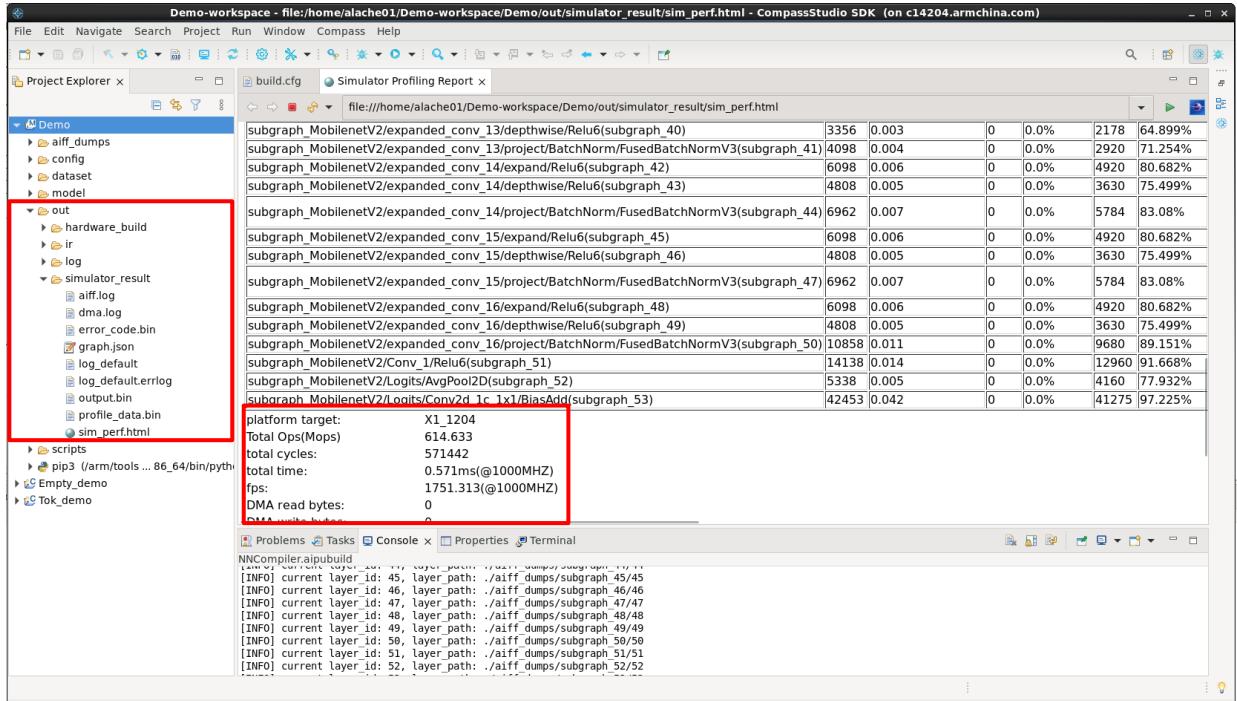
图 4-12: 打开 Profiler 工具



- 下面将以 X1 为例展示具体使用步骤：

选中工程，在主菜单栏单击 **Compass > Profiler Report > Simulator Profiler**，CompassStudio 将根据“simulator_result”目录下的“graph.json”和“profile_data.bin”文件自动生成“sim_perf.html”文件（如图 4-13 所示），并自动打开展示该文件。另外，若目录中已经存在“sim_perf.html”文件，将提示用户是否重新生成该文件；若不存在，CompassStudio 会生成“sim_perf.html”文件然后在主页面展示。

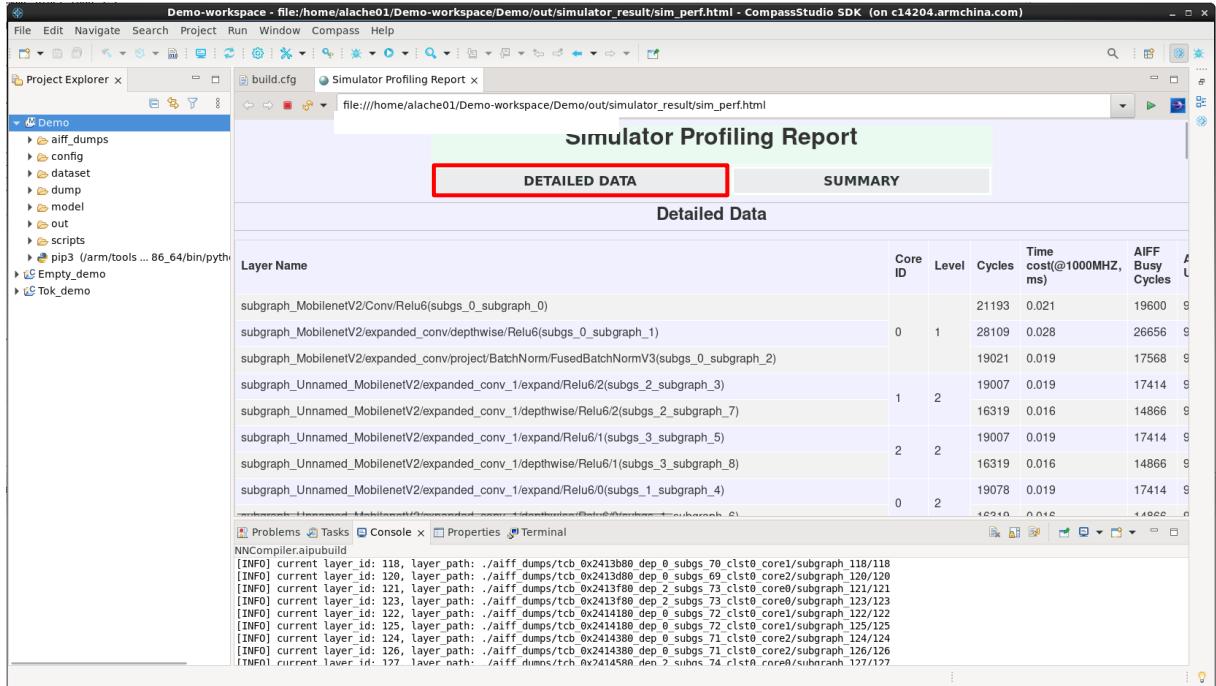
图 4-13: CompassStudio 生成 profiler 报告



- 下面将以 X2 为例展示 Simulator Profiler:

图 4-14 展示了在 X2_1204MP3 上运行 Simulator Profiler 的结果。

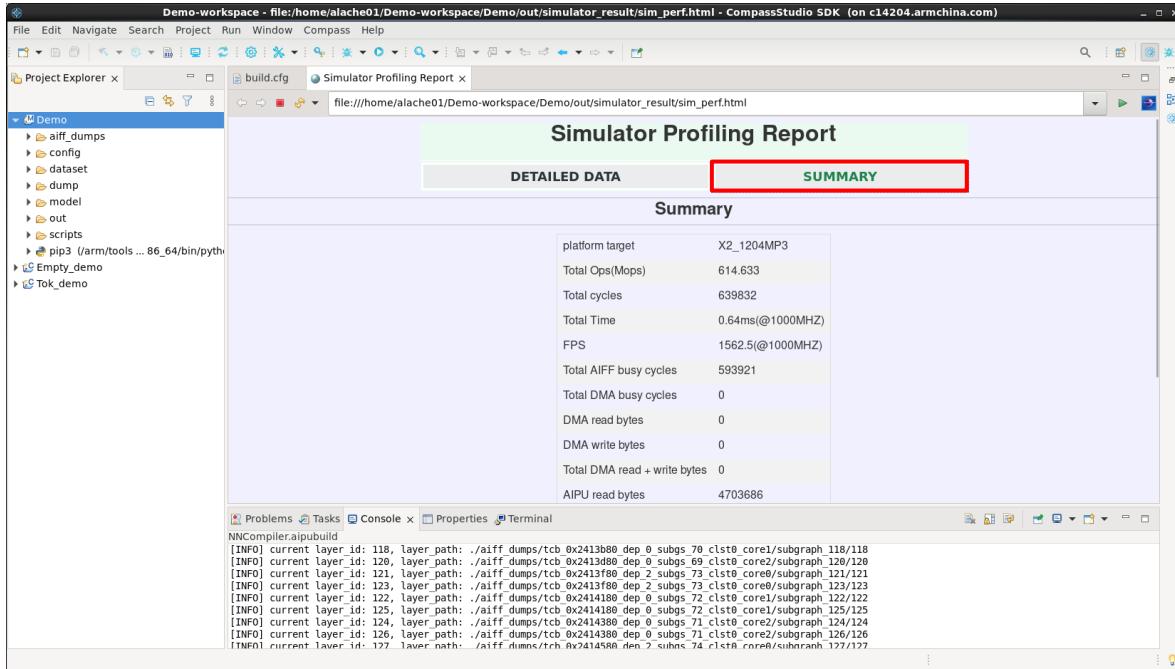
图 4-14: X2_1204MP3 上运行 Simulator Profiler



X2 系列的 Simulator Profiler 包含两个页签：

- “Detailed Data” 展示了各层的详细信息，如图 4-14 所示。
- “Summary” 展示了网络的总体信息，如图 4-15 所示。

图 4-15: X2 Simulator Profiler Summary 页签



4.6.2 Hardware Profiler

Hardware Profiler 执行步骤跟 Simulator Profiler 一致，请参考 4.6.1。在 AIPU 的开发板上执行 Run on Hardware (参考 4.3) 生成 buffer 文件 (“profiler_data.bin”)。在主菜单栏单击 Compass > Profiler > Hardware Profiler 将会生成 Hardware Profiler 可视化报告。

图 4-16 和图 4-17 以 X2_1204MP3 为例展示 Hardware Profiler。

图 4-16: X2_1204MP3 上运行 Hardware Profiler

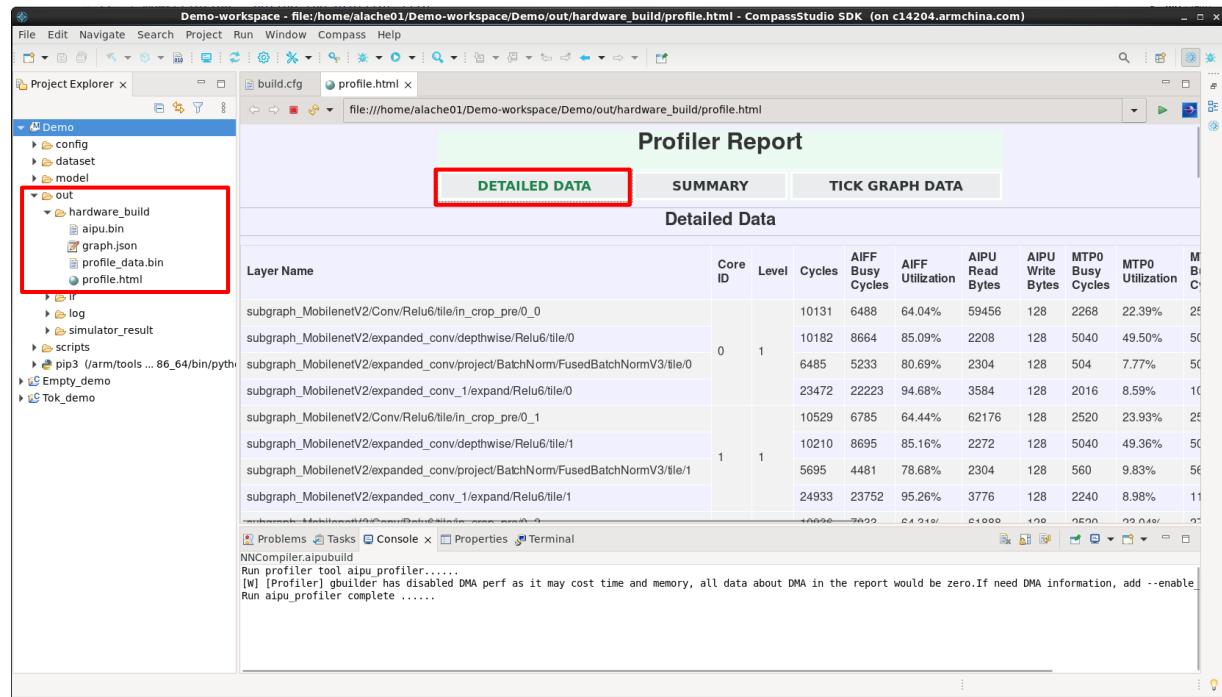
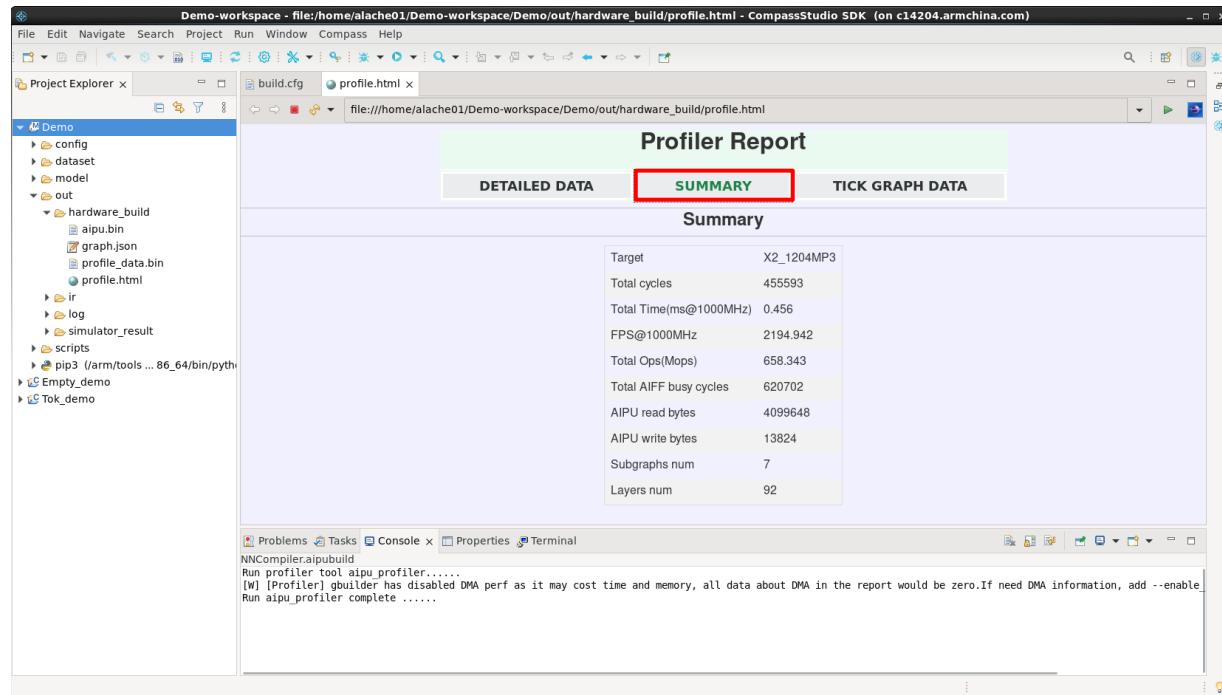


图 4-17: X2 Hardware Profiler Summary 页签



4.6.3 Multi-core profiler

目前该版本的 Multi-core profiler:

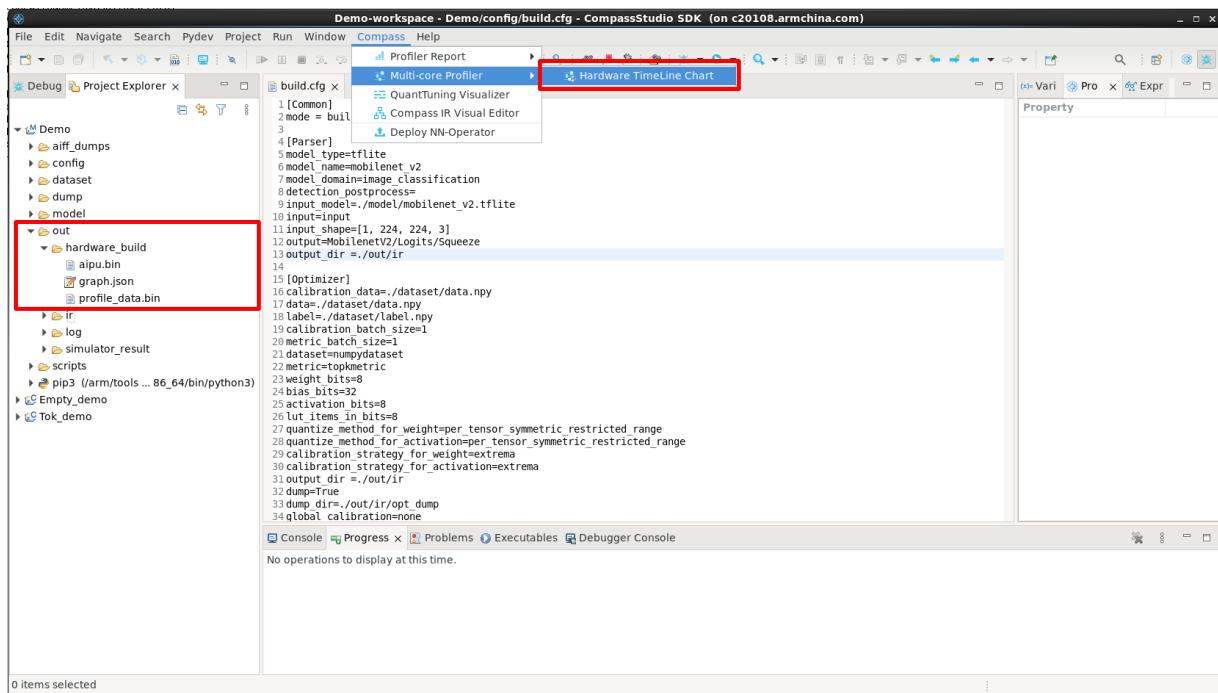
版权所有 © 2022–2023 安谋科技（中国）有限公司。保留所有权利。
保密信息

- 仅对 Compass X2_1204MP3 提供支持。
- 仅对 Hardware 提供支持。

Multi-core profiler 的使用步骤请参考：

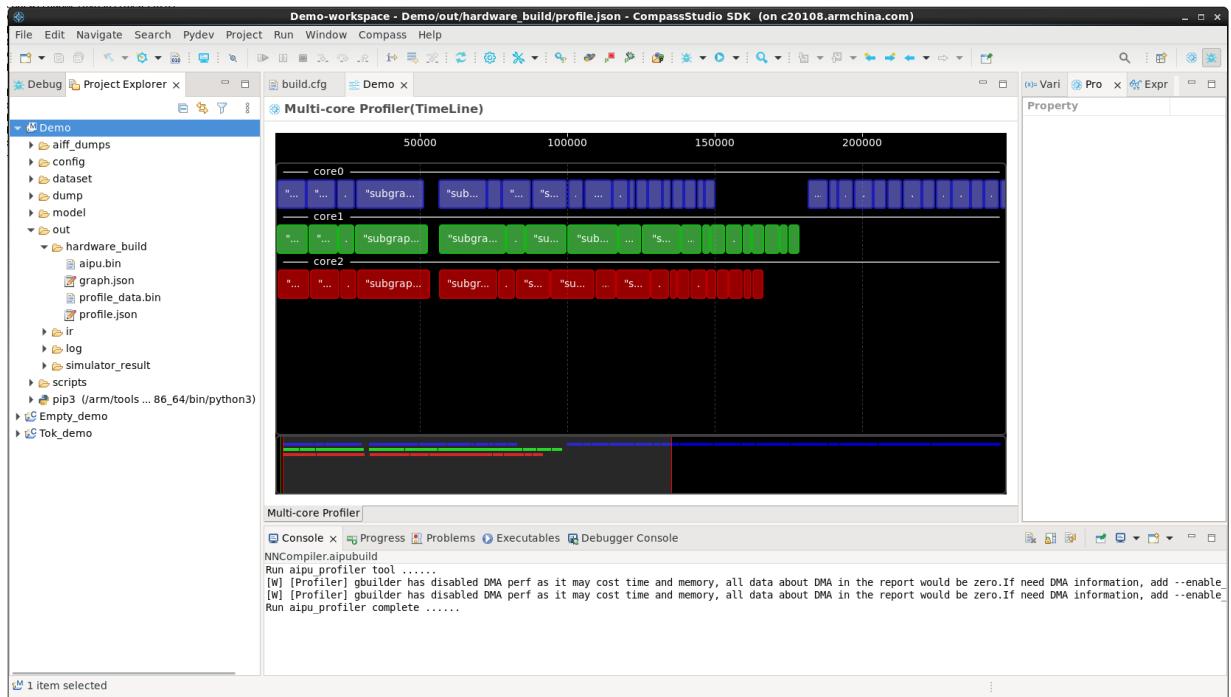
1. 将 Hardware 环境中 dump 出的 buff 文件重命名为“profile_data.bin”，并且放置在“/out/hardware_build”目录下。
2. 在主菜单栏单击 **Compass > Multi-core profiler > Hardware TimeLine Chart**，如图 4-18 所示。

图 4-18：执行 Multi-core profiler



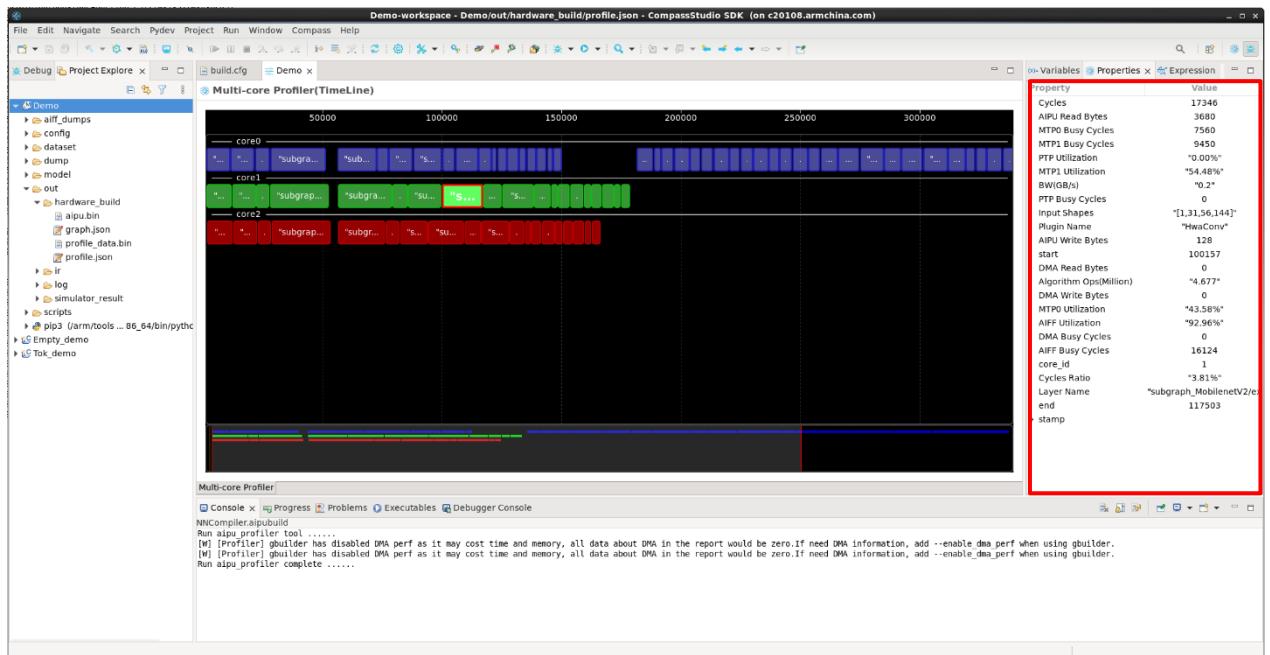
3. 执行完步骤 2，会打开 Multi-core profiler chart，如图 4-19 所示。

图 4-19: Multi-core profiler 视图



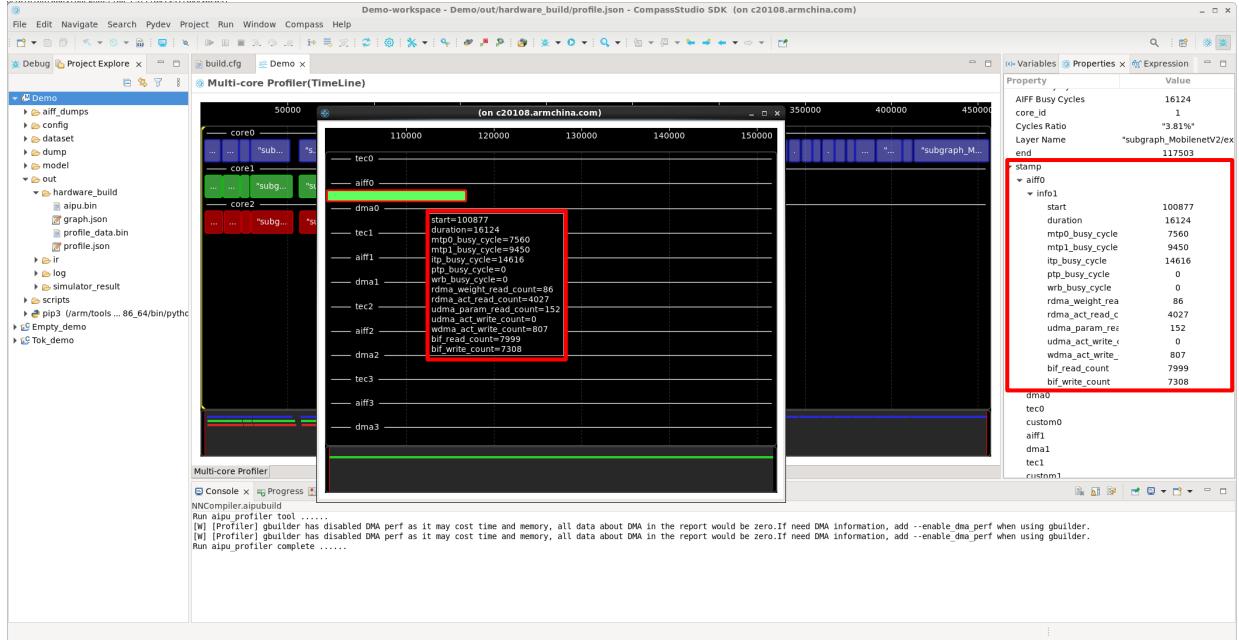
- 在图 4-19 中, Multi-core profiler 视图的横坐标是“Cycle”，会显示每个 subgraph 的 cycle 周期。纵坐标显示的是每个 core 下执行的 subgraph 。单击 subgraph 会在“Properties”视图中显示具体信息，如图 4-20 所示。

图 4-20: Subgraph 信息展示



5. 在图 4-20 中, 双击 subgraph, 会弹出 subgraph 中的 TEC/DMA/AIFF 信息, 如图 4-21 所示。

图 4-21: TEC/DMA/AIFF 信息展示



6. 在图 4-20 中, 将鼠标放置在 TEC/DMA/AIFF 的图表中, 会自动显示信息。您也可以在“Properties”视图中展开“stamp”查看 TEC/DMA/AIFF 的具体信息, 如图 4-21 所示。

4.7 高级功能

4.7.1 Compass IR Run 和 Build

本小节主要介绍使用 CompassStudio 将 Model 通过 NN-Model Parse 和 NN-Model OPT 生成 Float Compass IR 和 Quant (int8) Compass IR, 再使用 Compass IR Build/Run 功能产生可以在 Hardware/Simulator 上运行的 bin 文件。

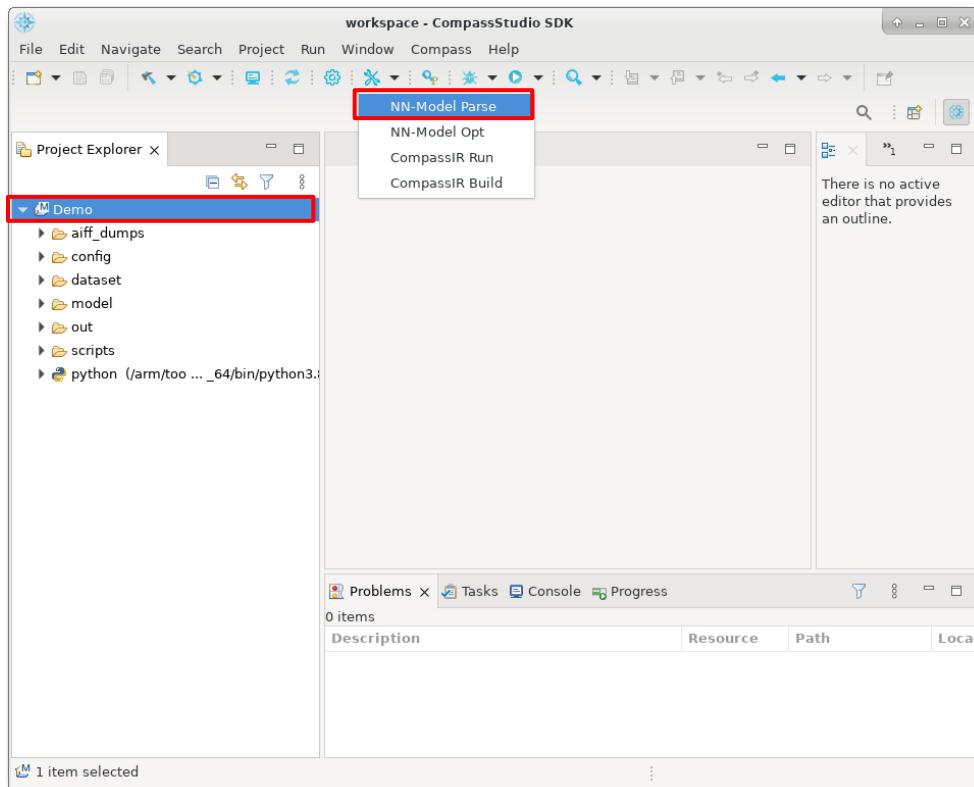
Compass IR Run 和 NN-Model Run 一样会直接运行在 Simulator 上, 并且产生“output.bin”。Compass IR Build 不会直接运行在 Hardware 上, 只会产生运行在 Hardware 上的 bin 文件。

NN-Model Parse

将 Model 生成为 Float Compass IR 的步骤如下:

- 选择工程，然后在工具栏单击 NN-Model Parse 图标，如图 4-22 所示。

图 4-22: 打开 NN-Model Parse



- 单击 NN-Model Parse，打开 aipuparse 运行界面，如图 4-23 所示。

图 4-23: NN-Model Parse 界面

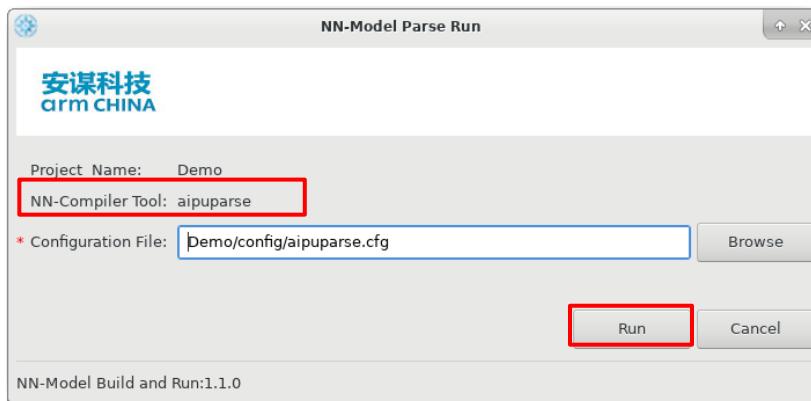


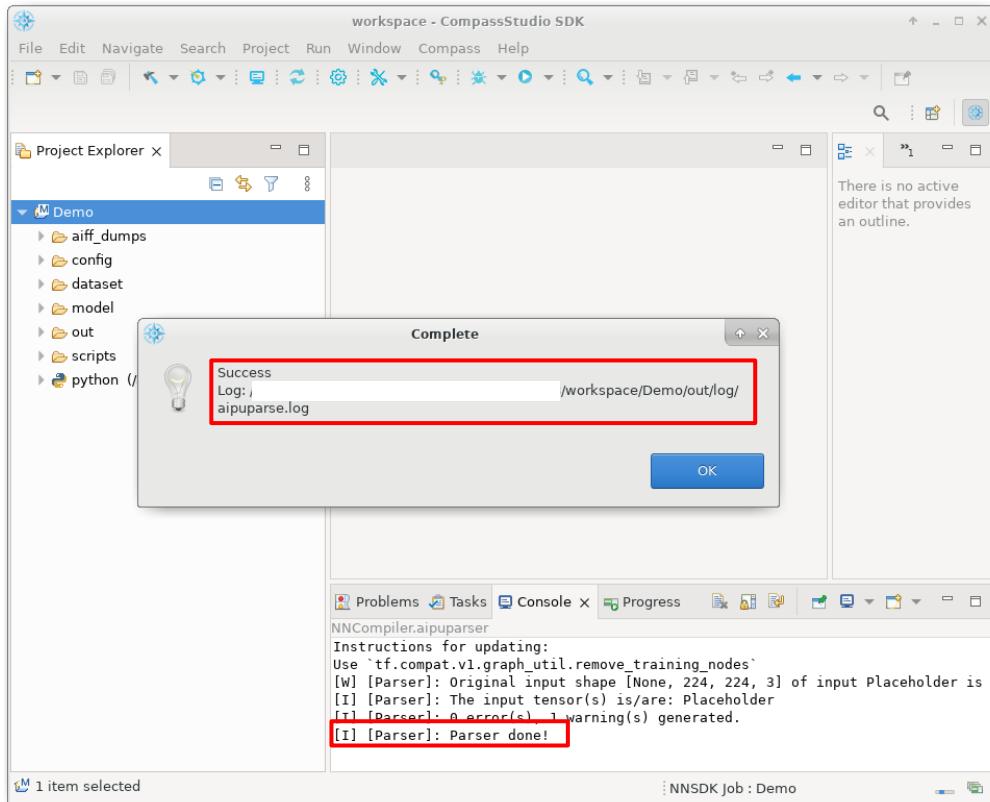
表 4-4: 参数描述

参数	描述
Project Name	选定的工程名称。

参数	描述
NN-Compiler Tool	NN-Compiler 工具, 该模式下默认是 aipuparse。
Configuration File	CompassStudio 会根据您选择的 Build 模式, 自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件, 单击 Browse , 然后选择自定义的 cfg 文件路径。

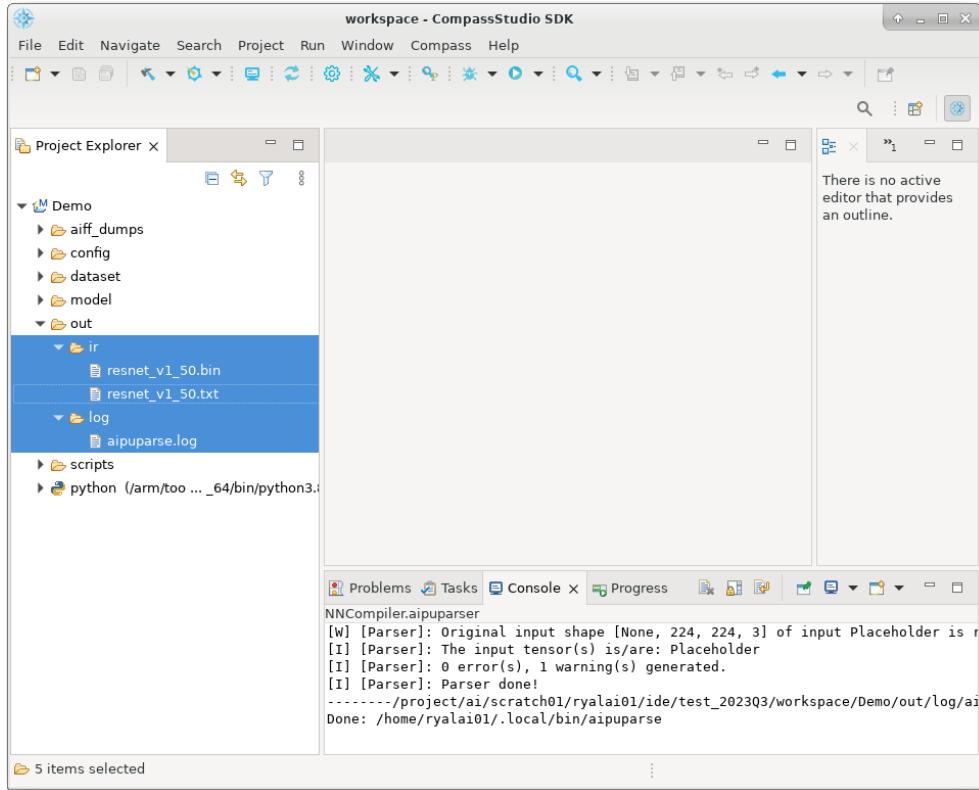
3. 配置完 NN-Model Parse 参数后, 单击 Run, 运行 NN-Model Parse。Console 页签会实时输出日志, 并且会弹出进度条, 如图 4-24 所示。

图 4-24: NN-Model Parse 运行结果



4. 运行结束后, 会弹出“Success”。您也可以在 Console 页签查看运行日志, 日志最后一行会显示运行 aipuparse run 的结果。“Parse done”表示运行结束。同时日志也会保存到工程的“out/log/aipuparser.log”文件。如果运行成功, 会在工程的“out/ ir”目录下生成相应的 Float IR, 如“xxx.bin”和“xxx.txt”文件。以 TensorFlow Lite 为例生成的文件为“mobilenet_v2.bin”和“mobilenet_v2.txt”文件, 如图 4-25 所示。

图 4-25: NN-Model Parse 结果



NN-Model Opt

CompassIR Quant 和优化的步骤如下:

1. 参照图 4-22, 选择 NN-Model Opt, 打开 aipuopt 运行界面, 如图 4-26 所示。

图 4-26: NN-Model Opt 界面

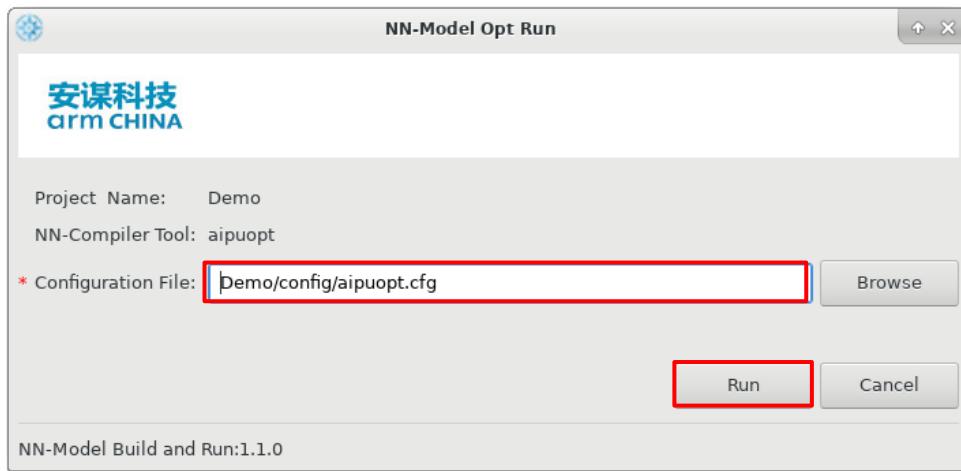
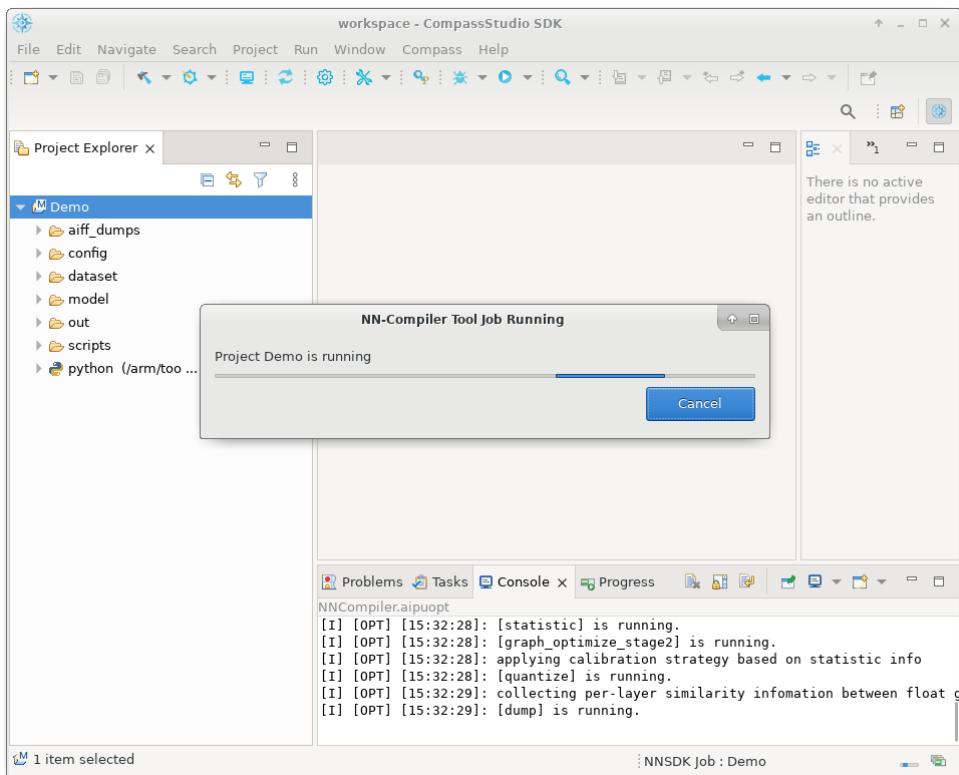


表 4-5: 参数描述

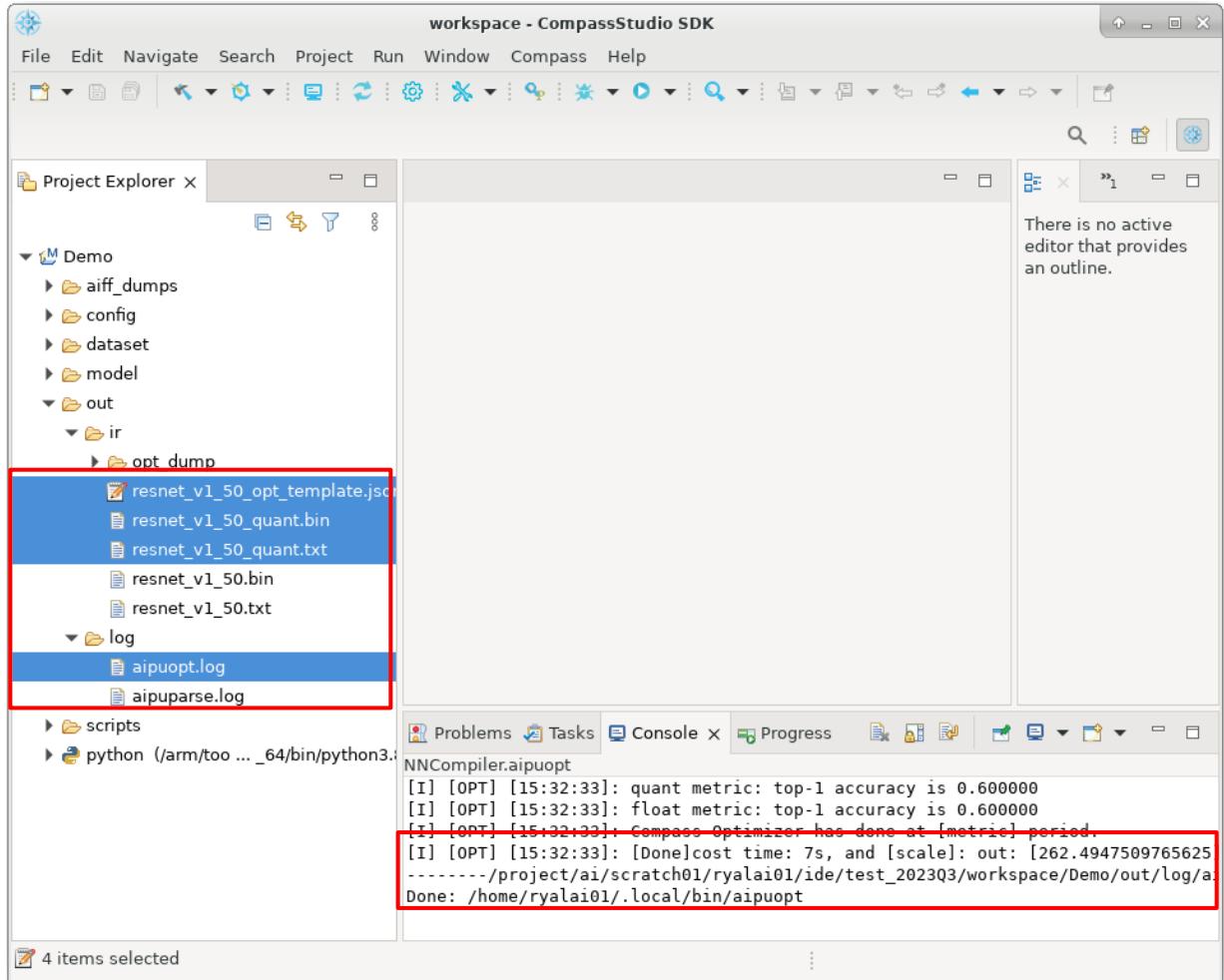
参数	描述
Project Name	选定的工程名称。
NN-Compiler Tool	NN-Compiler 工具, 该模式下默认是 aipuopt。
Configuration File	CompassStudio 会根据您选择的 Build 模式, 自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件, 单击 Browse, 然后选择自定义的 cfg 文件路径。

2. 配置完 NN-Model Opt 参数后, 单击 Run, 运行 NN-Model Opt。Console 页签会实时输出日志, 并且会弹出进度条, 如图 4-27 所示。

图 4-27: NN-Model Opt 运行中



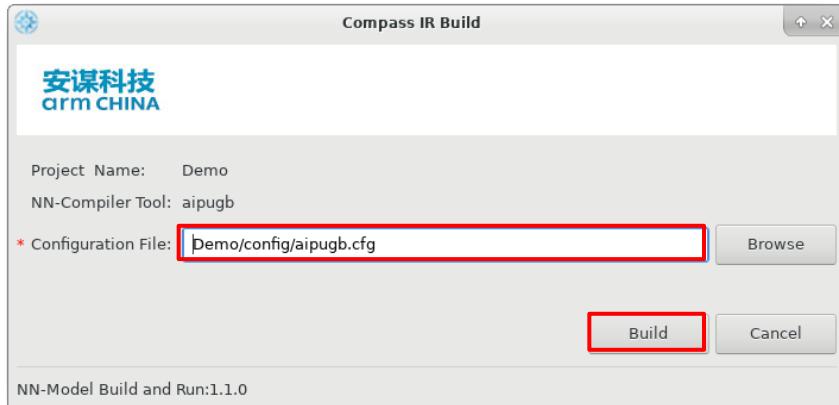
3. 运行结束后, 会弹出“Success”。您也可以在 Console 页签查看日志, 日志最后一行会显示运行 aipuopt run 的结果。“Opt done”表示运行结束。同时日志也会保存到工程的“out/log/aipuopt.log”文件。如果运行成功, 会在工程的“out/ir”目录下生成 Quant IR。以 TensorFlow Lite 模型为例, 会生成“mobilenet_v2_opt_template.json”、“mobilenet_v2_quant.bin”、“mobilenet_v2_quant.txt”和“mobilenet_v2_statistic_info.npy”文件, 如图 4-28 所示。

图 4-28: NN-Model Opt 结果

Compass IR Build

将 Compass IR 生成到“Hardware.bin”的步骤如下：

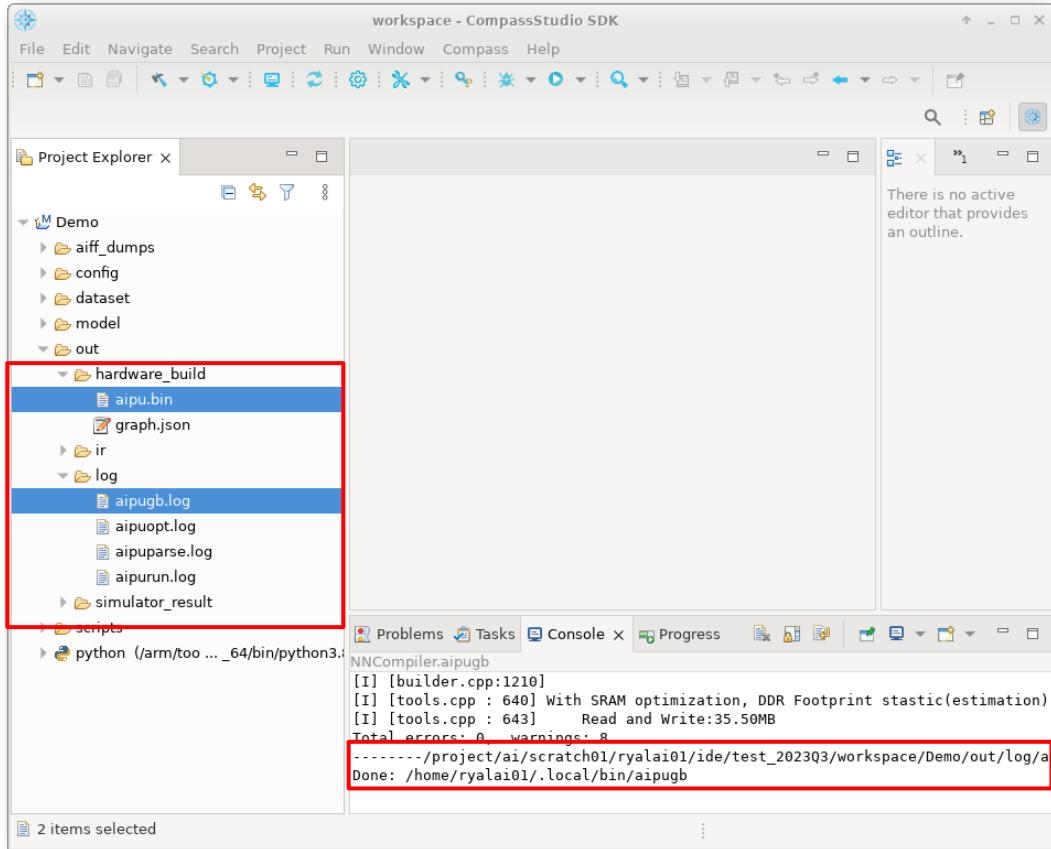
1. 参照图 4-22，选择 CompassIR Build，打开 aipugb Build 界面，如图 4-29 所示。

图 4-29: Compass IR Build 界面**表 4-6: 参数描述**

参数	描述
Project Name	选定的工程名称。
NN-Compiler Tool	NN-Compiler 工具, 该模式下默认是 aipugb。
Configuration File	CompassStudio 会根据您选择的 Build 模式, 自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件, 单击 Browse, 然后选择自定义的 cfg 文件路径。

- 配置完 Compass IR Build 参数后, 单击 **Build**, 运行 Compass IR Build。Console 页签会实时输出日志, 并且会弹出进度条, 运行结果如图 4-30 所示。

图 4-30: Compass IR Build 结果



- 运行结束后，可在 Console 页签查看日志，日志最后一行会显示 aipugb Build 的运行结果。“Total error:0, warning:1”表示运行结束，且有 0 个错误，1 个警告信息。同时日志也会保存到工程的“out/log/aipugb.log”文件中。正常运行结束后，会在工程的“out/hardware_build”目录下生成的可执行文件“aipu.bin”。如果“hardware_build”文件夹已经存在，将会被当前输出结果覆盖。

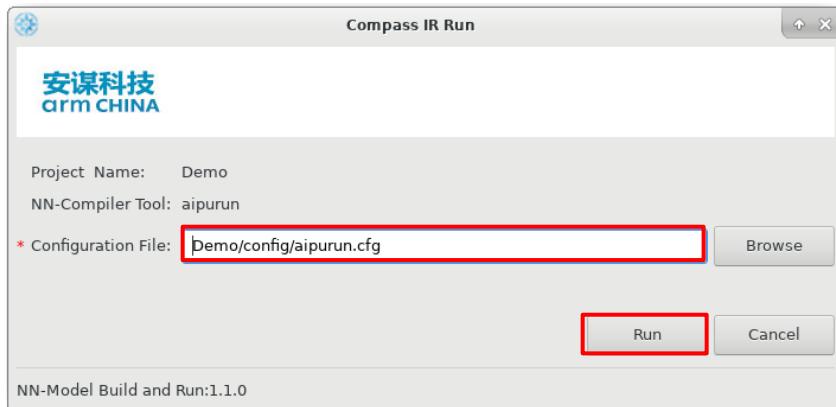
Compass IR Run on Hardware

请参考 [4.3 Run on Hardware](#)。

Compass IR Run on Simulator

如 [4.4](#) 章节所述，Compass IR Run 和 NN-Model Run 一样，都是运行 Run on Simulator，具体步骤如下：

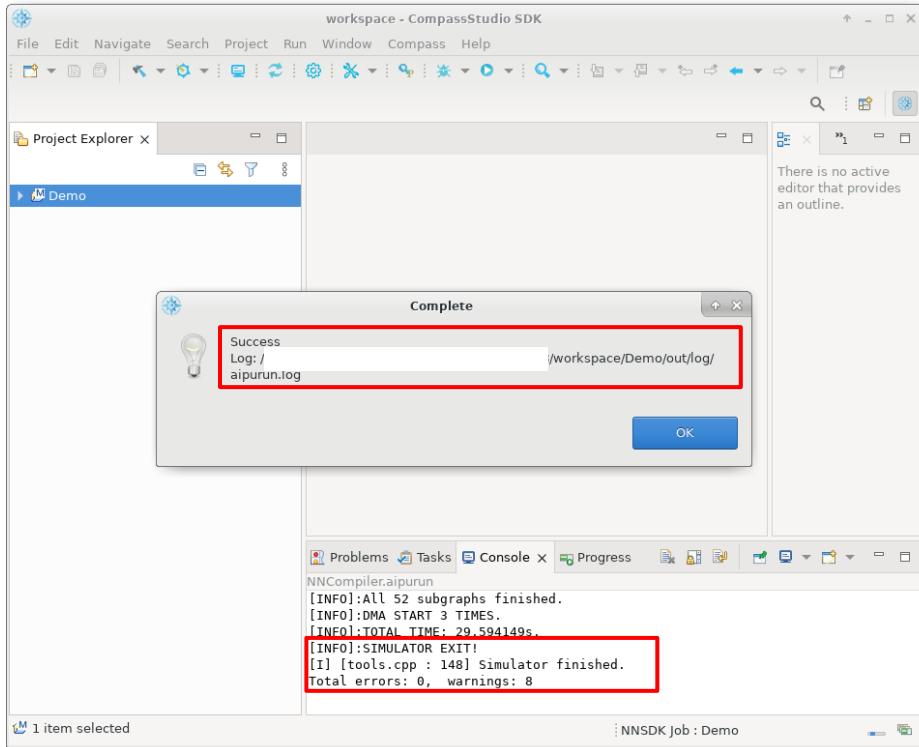
- 参照图 4-22，选择 CompassIR Run，打开 aipuprun 运行界面，如图 4-31 所示。

图 4-31: Compass IR Run 界面**表 4-7: 参数描述**

参数	描述
Project Name	选定的工程名称。
NN-Compiler Tool	NN-Compiler 工具, 该模式下默认是 aipurun。
Configuration File	CompassStudio 会根据您选择的 Build 模式, 自动加载工程下的 cfg 文件。要想使用自定义的 cfg 文件, 单击 Browse , 然后选择自定义的 cfg 文件路径。

2. 配置完 Compass IR Run 参数后, 单击 **Run**, 运行 Compass IR Run。Console 页签会实时输出日志, 并且会弹出进度条, 运行结果如图 4-32 所示。

图 4-32: Compass IR Run 结果



- 运行结束后，可在 Console 页签查看日志，日志最后一行会显示 aipurun 的运行结果。“Total error:0, warning:2”表示运行结束，且有 0 个错误，2 个警告信息。同时日志也会保存到工程的“out/log/aipurun.log”文件中。正常运行结束后，会在工程的“out”目录下生成目录文件“aipurun_result”。该文件夹下存放 aipurun 运行的结果文件。如果“simulator_result”文件夹已经存在，将会被当前输出结果覆盖。

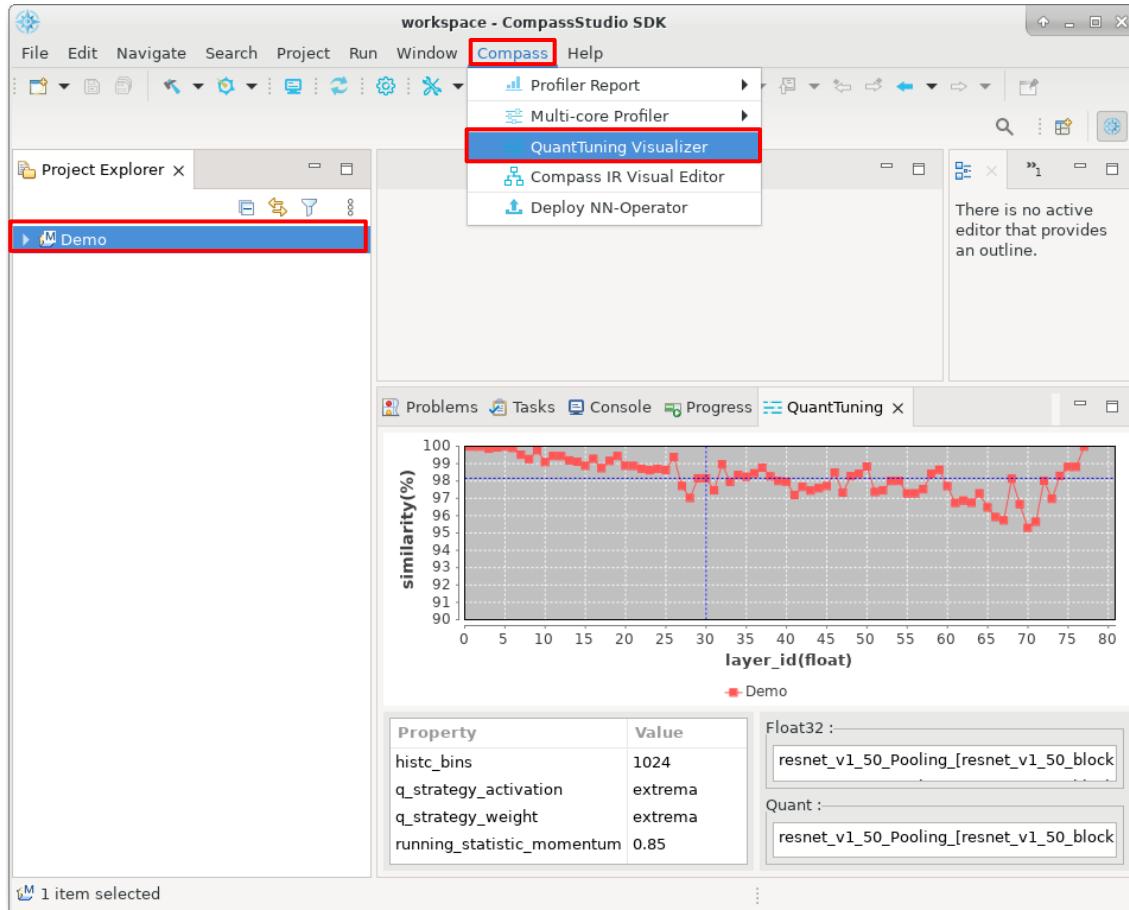
4.7.2 QuantTuning

QuantTuning 会显示 OPT 阶段 quant 调优的信息以及整个网络相似度趋势。在 QuantTuning 趋势图中，可以选择任一节点，查看该节点对应 layer 的基本信息和调优参数信息。通过观察 QuantTuning 趋势图的走向，对相似度较低的节点进行信息查看，并可以通过 QuantTuning 趋势图打开对应节点的 CompassIR Visualizer 视图且可以修改 Optimizer 的参数，修改完成后保存 CompassIR Visualizer。再次运行 NN-Model Build 和 Run，此时打开 QuantTuning 就会显示调优后的网络相似度趋势。

该模块需要用户先成功运行 aipuparse Run 和 aipuopt Run 产生 Float IR 和 Quant IR 相关数据（请参考 [NN-Model Parse](#) 和 [NN-Model Opt](#)）。如果没有执行上述步骤就使用该功能，CompassStudio 会在 Console 中提示相关错误信息。以下步骤默认您已经成功运行 aipuparse Run 和 aipuopt Run。

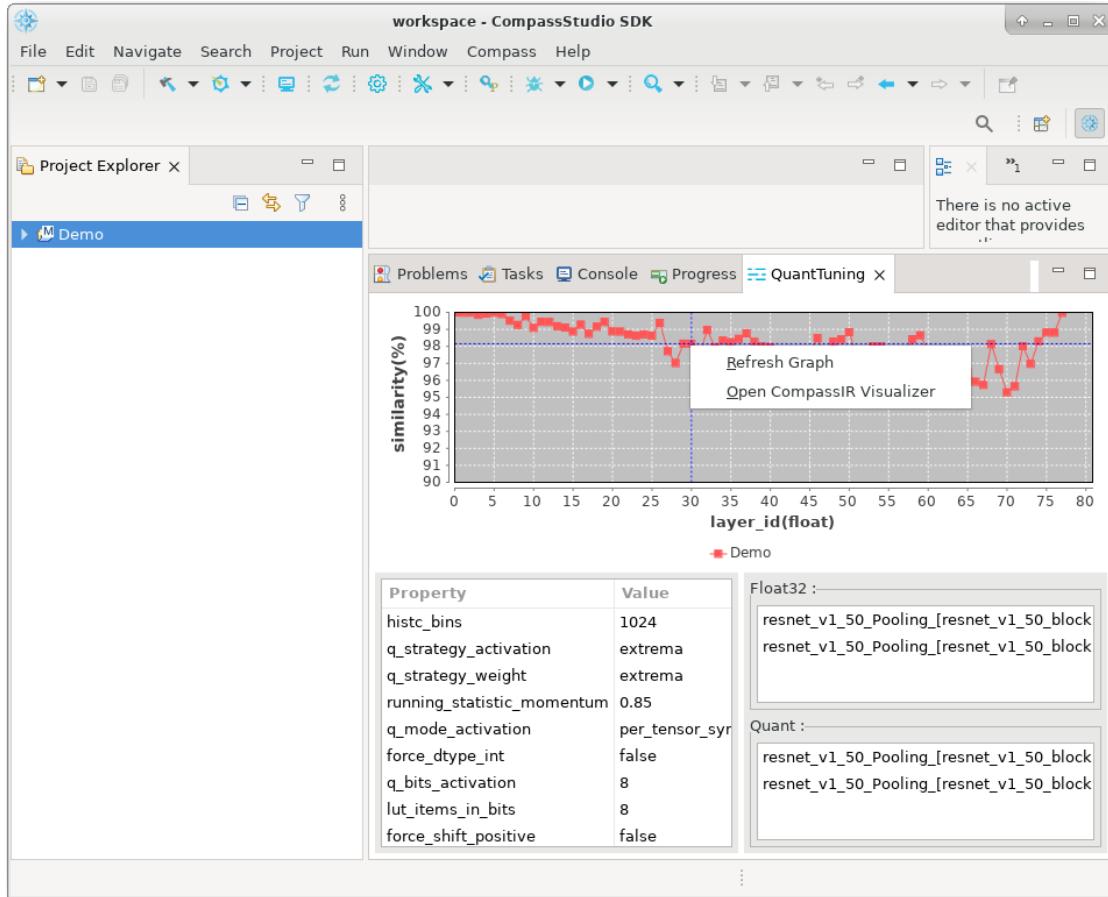
1. 选中工程，然后单击菜单 **Compass > QuantTuning Visualizer**，显示 Graph，如图 4-33 所示。

图 4-33: 网络 Graph

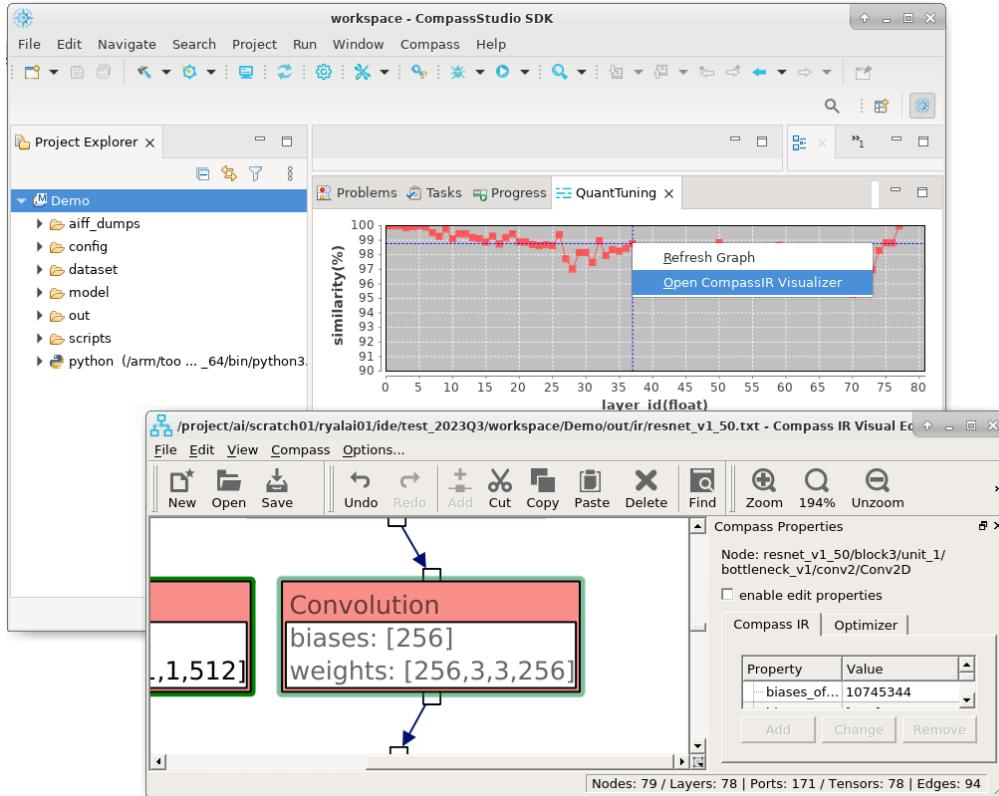


该 Graph 可以显示模型的整个网络信息，您可以观察网络中 layer_id 的趋势，从而发现异常。

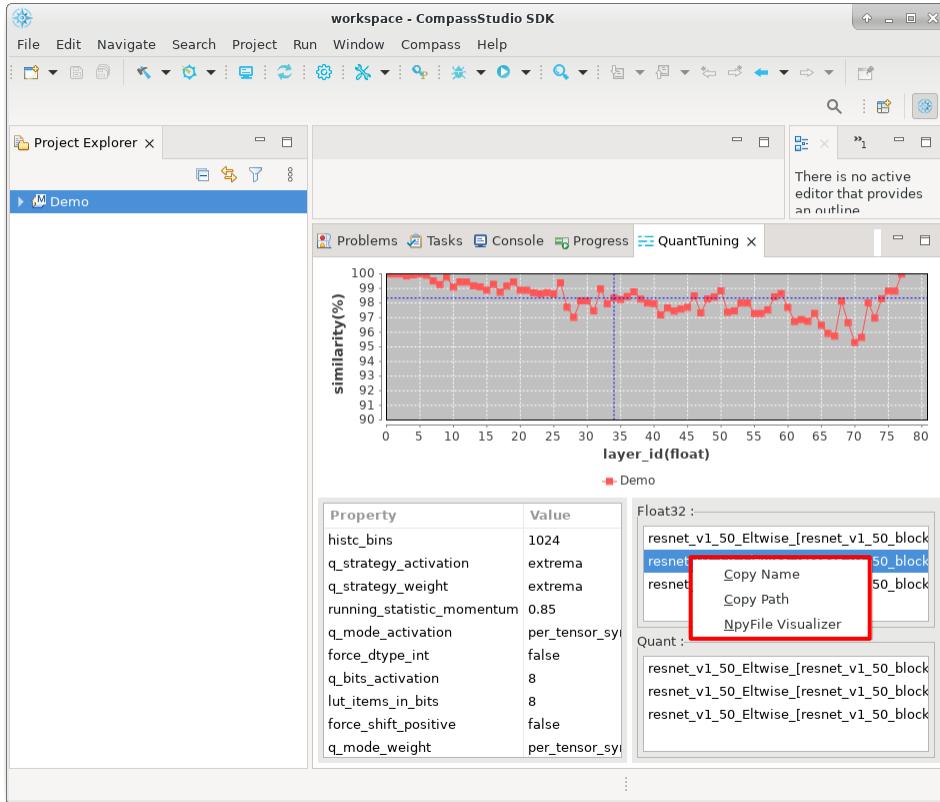
2. 在 Graph 中，将鼠标指针悬停在任一节点上，可以显示该节点的 X、Y 坐标。单击该节点，会显示该 layer_id 的 quant 信息，并且会显示和该 layer_id 对应的“dump npy”(float32、quant)文件，如图 4-34 所示。

图 4-34: QuantTuning 视图

3. 右键单击 Graph 图中的任一位置，然后单击 Refresh Graph 可以刷新 Graph。右键单击任意节点，然后单击 Open CompassIR Visualizer，可以打开模型可视化图，并且在模型可视化图上聚焦到 QuantTuning 视图当前节点对应的 layer 上，如图 4-35 所示。

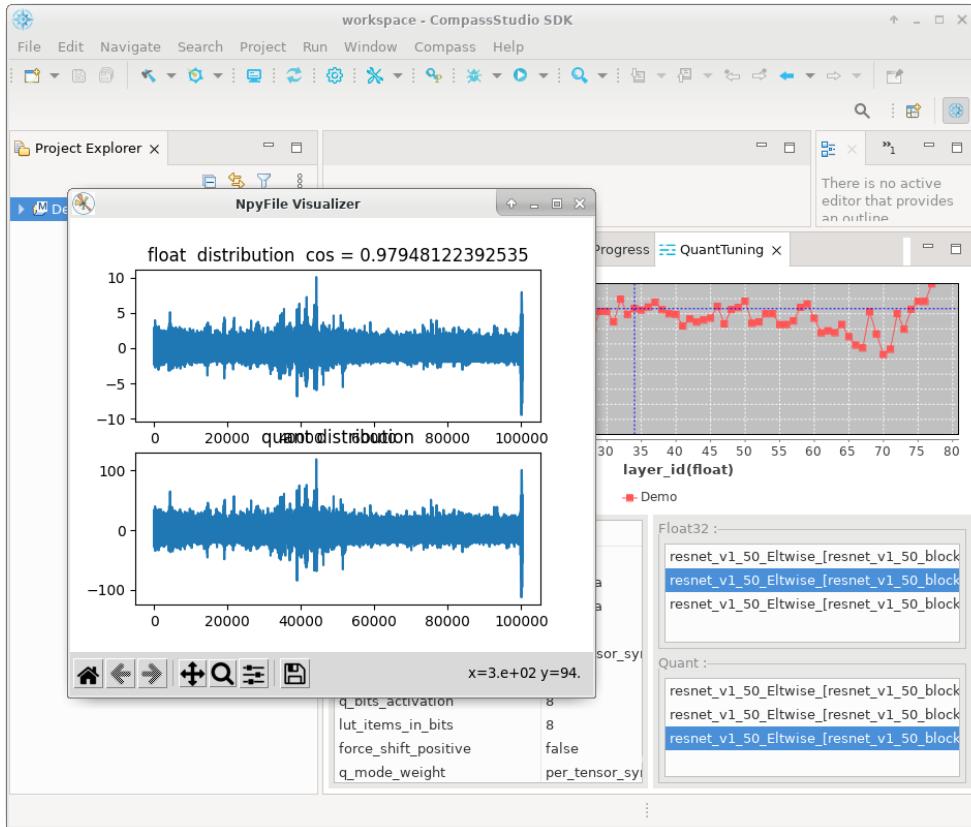
图 4-35: QuantTuning Graph 功能

4. 对于图 4-35 中的 quant table 信息，可以查看 quant 信息。对于 dump 文件，右键单击文件，可以选择 Copy Name、Copy Path、NpyFile Visualizer，如图 4-36 所示。

图 4-36: QuantTuning 信息展示

5. 在图 4-36 中，在任意 Group 中选择 dump 出来的 NpyFile 文件（CompassStudio 会根据选择的 NpyFileName 去查找对应的 Float32 和 Quant NpyFile 进行可视化比较），单击 NpyFile Visualizer，进行 Npy 文件可视化展示，如图 4-37 所示。

图 4-37: Npy 文件可视化展示



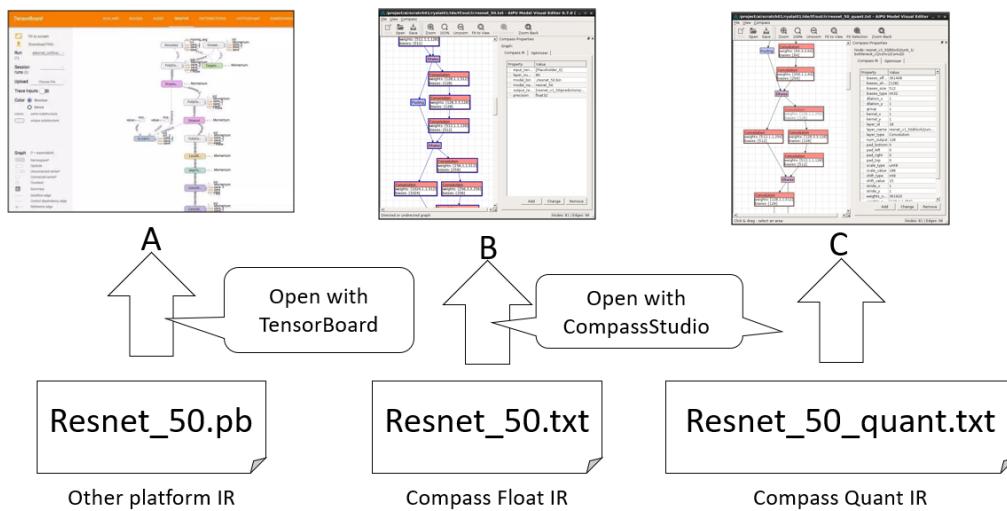
- 对于 Graph 的大小，可以使用 **ctrl + 鼠标滚轮** 来放大或缩小，这种情况下是对整个 Graph 进行放大或缩小。要想以节点为中心进行放大或缩小，可以先将鼠标光标悬停在某一节点上，然后使用 **ctrl + 鼠标滚轮** 来放大或缩小。

4.7.3 Compass IR Visual Editor

在模型评估阶段，可以通过 CompassStudio 内置的可视化工具 Compass IR Visual Editor 预览 Compass SDK 解析的图与预期的图结构是否一致。与大多数 AI 网络可视化工具不同的是，Compass IR Model Visual Editor 除了可以将图的拓扑结构渲染为二维图像，还可以直接在可视化工具中修改节点属性和量化属性。

以模板中的 TensorFlow resnet_50 为例，通过对比图 4-38 中的图 A（由对应平台的可视化工具 tensorboard/netron 生成）和图 B（Float IR），评估您的网络是否解析正确。通过对比图 B（Float IR）和图 C（Quant IR），评估您的网络量化结果是否合适。

图 4-38: 使用模型可视化工具评估网络结构



启动模型可视化工具

可以通过下表中的方法打开可视化工具。

表 4-8: 启动模型可视化工具

启动方法	操作	功能	图例
打开单个 IR 文件	在 Project Explorer 列表中, 选中工程, 选择 out > ir , 然后右键单击 “your_ir.txt”, 选择 Compass IR Visual Editor 。	打开单个 IR。如果同文件夹有同名量化参数配置文件, 会自动加载量化参数。	图 4-39
打开工程默认 IR	单击 Compass > Compass IR Visual Editor 菜单。	从工程的缺省位置加载 IR。	图 4-40
量化调优界面选择单个节点	在 QuantTuning 页签, 右键单击节点, 然后选择 Open ModelVisualizer 。	打开单个 IR, 并聚焦至所选节点。	图 4-41

图 4-39: 打开单个 IR

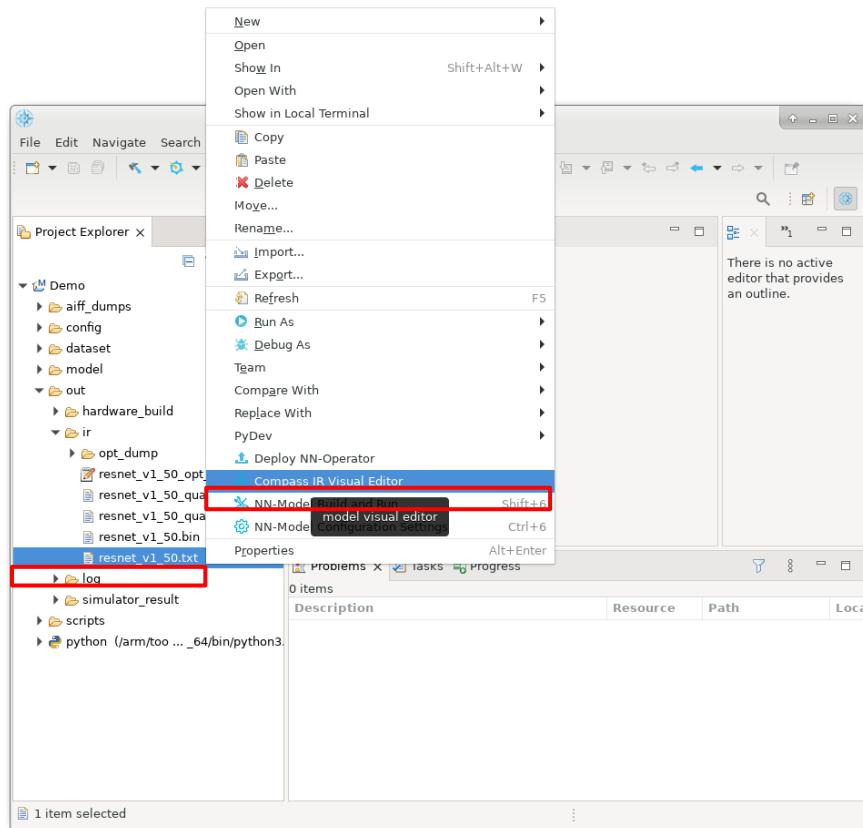


图 4-40: 打开工程默认 IR

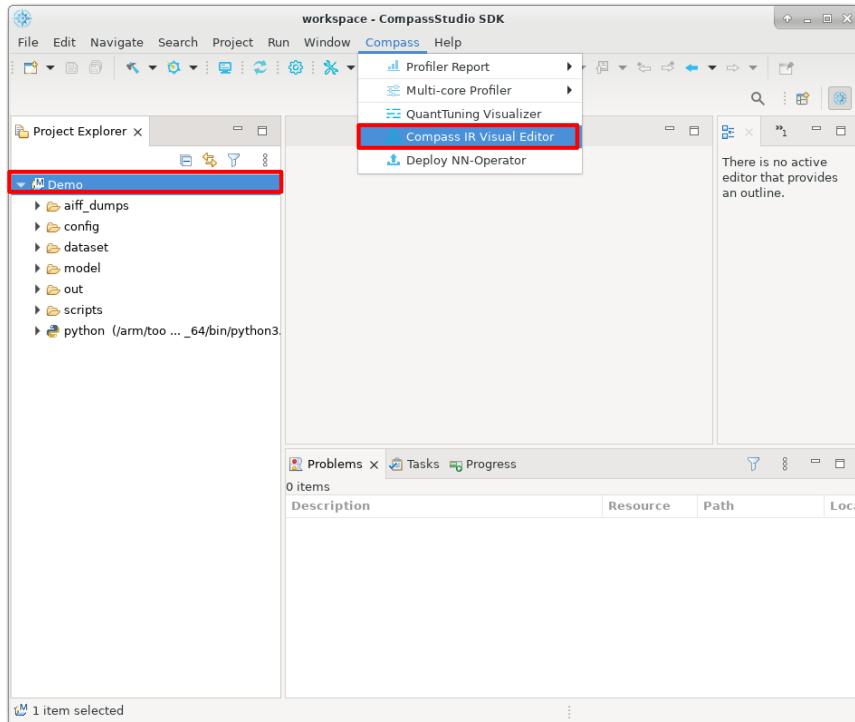
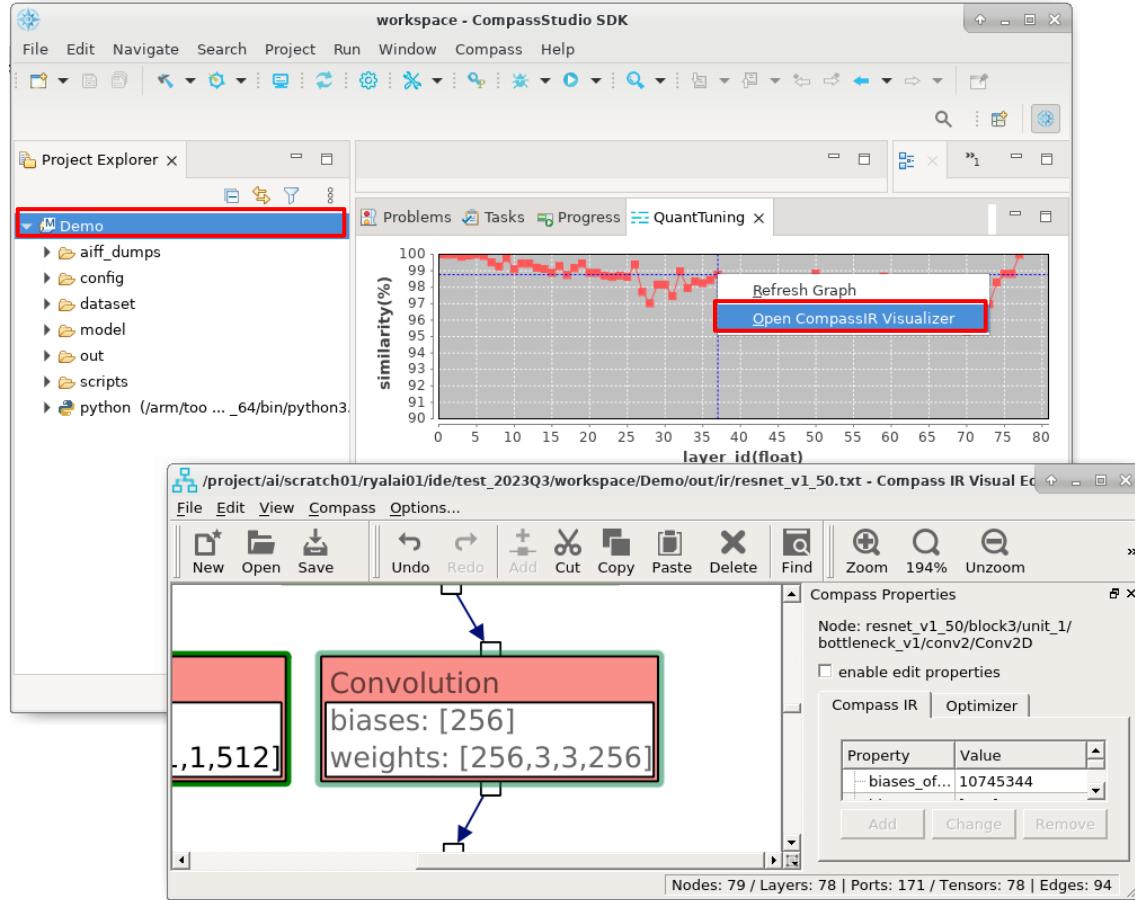


图 4-41: 在量化调优界面选择单个节点



功能介绍

NN-Model Visual Editor 的主界面如图 4-42 所示，包括以下几个功能区域：

- 菜单和工具栏
- 网络结构 Graph 视图
- Compass IR 属性编辑区

图 4-42: 功能区域



有关各区域的具体功能，请参考下表。

表 4-9: 功能介绍

区域	操作 (*在工具栏有快捷入口)	功能
菜单和工具栏	File > Open *	从资源管理器打开一个 IR。(如果 IR 不在 Eclipse project 下)
	File > Save *	保存对 IR 的修改
	File > Save as ...	IR 文件另存为
	File > Export to img ...	导出拓扑结构图到图片
	File > Recent File	打开最近浏览的 IR
	File > Close	关闭窗口
	Edit -> Undo *	撤销
	Edit -> Redo *	重做
	Edit -> Add	新建节点
	Edit -> Cut *	剪切选区项目
	Edit -> Copy *	复制选区项目
	Edit -> Paste *	粘贴选区项目
	Edit -> Delete *	删除选区项目
	Edit -> Find *	查找选区项目
	View > Show Grid	是否打开背景网格
	View > Show Node Ids	是否显示节点的 ID

区域	操作 (*在工具栏有快捷入口)	功能
	View > Show Tensor Shapes of Node	是否显示节点内张量的形状
	View > Show Edge Ids	是否显示边的 ID
	View > Show Tensor of Edge	是否显示边对应张量的形状
	View > Zoom/UnZoom/Reset Zoom/Fit to View/Zoom Back *	调整视图大小
	View > Toolbars and Panels	是否开启编辑区
	View > Layout(GraphViz dot)	重新布局
	Compass > Check (built-in)	检查图结构
	Compass > Check (aipu_checker)	执行 aipu_checker 检查 IR 内容
	Compass > Attach Optimizer config file ...	加载量化参数文件
	Compass > Detach Optimizer config file	卸载量化参数文件
	Options...	选项设置
Graph 视图	单击节点	选择节点，编辑区绑定此节点信息
	单击边	选择边，编辑区绑定此节点信息
	单击空白处	选择图，编辑区绑定图信息
	Ctrl + 鼠标滚轮	调整显示区域
Compass IR 属性编辑区	选择属性，然后单击 Add/Change/Remove	编辑 Compass IR 属性（参考图 4-50）
提示和统计信息	-	提示内容以及统计图中节点/边的数目

可视化内容与 IR 对应关系

为了避免歧义，以下名词在本节中约定如下：

- **Node** 代表绘图中的节点
- **Port** 表示绘图中的端口
- **Edge** 表示绘图中的边
- **Layer** 代表 Compass IR 中的层/节点/算子
- **Tensor** 表示 Compass IR 中的张量

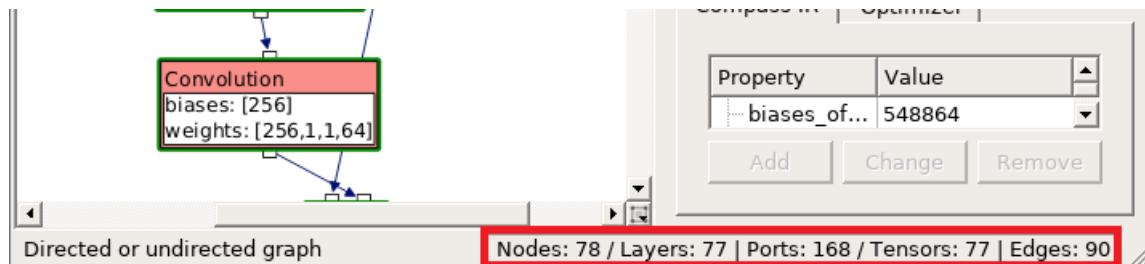
它们之间存在以下映射关系：

- 一个 Layer 绘制成为一个 Node。同时，为了展示效果，绘图中会出现 IR 文本中不存在 Layer 的情况，例如 Output Layer。
- 一个 Tensor 绘制成为三个部分，任意部分的操作都会传递到所属的 Tensor 上。
 - 一个 Out Port (节点下方的端口)
 - 一条或多条 Edge

- 一个或多个 In Port (节点上方的端口) 。

窗口右下角的计数器会实时统计图中上述项目的个数：

图 4-43：统计区域



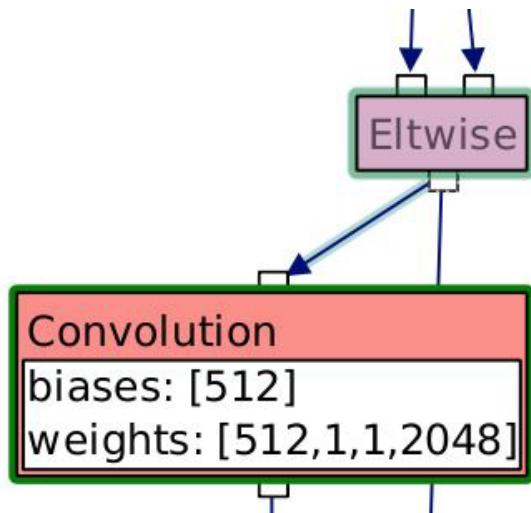
选择项目和选区

通过鼠标操作将图中项目 (Item) 放入选区，选中项目可以进行修改内容、删除、复制等操作。选择项目的操作方法如下：

- 区域选择：按住鼠标左键不放，拖动鼠标使用矩形框选择
- 单个选择：使用鼠标左键单击项目
- 单个增量选择：Ctrl + 鼠标左键单击项目
- 通过查找选择

被选中的 Node 和 Edge 亮度会被调高。被选中的 Port 轮廓会变成虚线。

图 4-44：选中与未选中的效果



例如上图中 Eltwise 节点，Eltwise 的输出端口以及与 Convolution 的一条边为被选中状态。

可以进入被选中的项目有：

- 图 Graph (点击空白处被视为选择图的操作)
- 节点 Node，包括 IR 中不存在的辅助节点
- 节点的端口 Port
- 边 Edge

不可以被选中的项目：

- 文本信息

新建节点

空白节点 Layer_type 为 Untitled。大多数情况下，您需要根据 IR 规范在 Compass Properties 窗口填入其他信息，才能保证这是一个合理的 Compass IR Layer。

以下任意一种方法可以新建一个空白节点，通过鼠标会在光标位置新建节点，其他方法会在图的左上方新建节点。

- 鼠标右键 > Add node
- 侧边栏 Edit > Add node
- 导航栏 Add node 图标

删除节点和边

选区有效的情况下，以下任意一种方法可以删除选区内容：

- 光标位置右键 > Delete
- 侧边栏 Edit > Delete
- 导航栏 Delete 图标

删除 Node 后，与其相连的 Edge 会被一同删除，所属的 Port 也会一同删除。选区内有 Port 但未选择其所属 Node 的情况，不会删除该 Port。

删除 Edge 后，与其相连的 Node 会保留。

复制粘贴

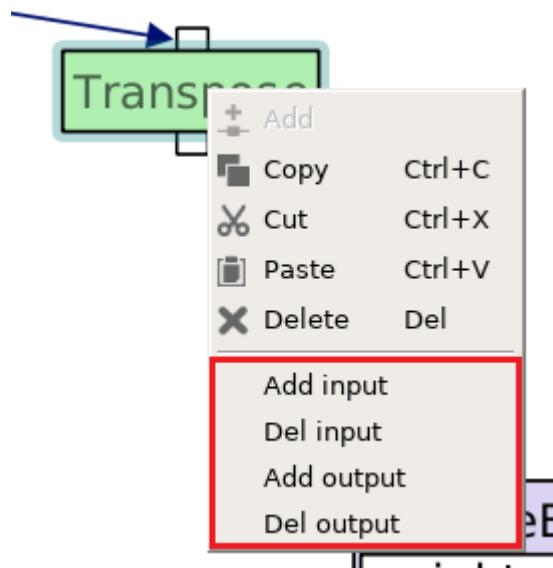
选区内的内容可以通过菜单栏或者右键的 Copy / Cut / Paste 菜单操作系统粘贴板。要注意的是，项目粘贴的时候，孤立的 Edge 和 Port 会被自动删除。如果粘贴到其他应用，会以图片格式传递。

编辑端口

在节点上方单击鼠标右键会弹出端口编辑菜单。可以对输入和输出端口分别操作：

- Add input: 追加一个输入端口
- Del input: 删除最后一个输入端口
- Add output: 追加一个输出端口
- Del output: 删除最后一个输出端口

图 4-45: 编辑端口



一个节点的张量信息始终保存在输出端口上，而输入端口不携带任何信息。删除端口后，与其链接的边会一同删除。对于输出端口，其属性会一同删除。

编辑图/节点/边属性

在 Compass 属性区域，可以直接编辑 Graph/Layer/Tensor 的属性，其中 Compass IR 页签为 Compass IR 的各个属性。如果您编辑的是 Float IR 且同时加载了量化参数时，可以在 Optimizer 页签编辑量化参数属性。

根据选区的内容，可编辑的目标按照以下规则随之改变：

- 选区为空：编辑 Graph 属性
- 选择单个 Node：编辑 Layer 属性
- 选择单个 Edge、Input Port、Out Port：编辑对应的 Tensor 属性
- 选区内有多个项目：不可编辑状态

在修改 Compass IR 和量化参数前, 请仔细查阅《 Zhouyi Compass IR Definition Application Note 》中要修改 Graph/Node/Tensor 的含义, 特别是规格说明中的支持范围。具体的量化策略可以参考《 周易 AIPU 量化算法白皮书 》和《 Arm China Zhouyi Compass Software Technical Overview 》。

连接节点

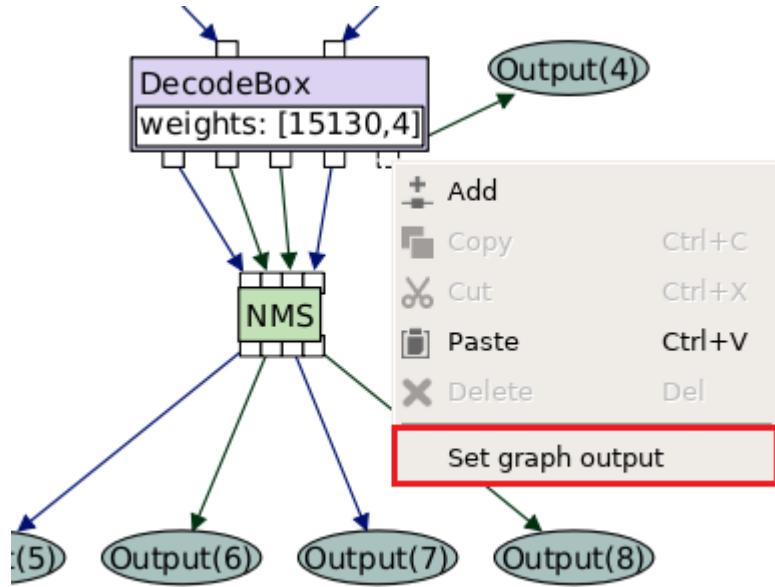
在节点的输出端口上按住鼠标左键, 拖动后会产生一个圆形节点。将圆形节点拖动至其他节点的输入端口, 即可完成节点的连接。

连接操作会限制一些行为来保证图的拓扑结构合理性, 例如反向连接或者连接自身。但是, 仍然会出现一些情况导致 Compass IR 无法支持, 例如出现了环路或者重名元素。这些情况在保存时会触发自动测试机制来避免。

编辑图的输入输出

在 Compass IR 定义中, 图的输入输出分别保存在 IR 的 `input_tensors` 和 `output_tensors` 的数组里, 对应 Input Node 和 Output Node。它们在数组中的索引显示在 `layer_type` 后的括号里, 并可以在对应 Tensor 的 `input_tensor_index` 和 `output_tensor_index` 字段里修改。

图 4-46: 设置图的输出张量



注意, `input_tensor` 需要有一个 `layer_type = Input` 的 Layer, 而 `output_tensor` 在 IR 里不需要 Output layer, 所以二者的新建方法有所不同。

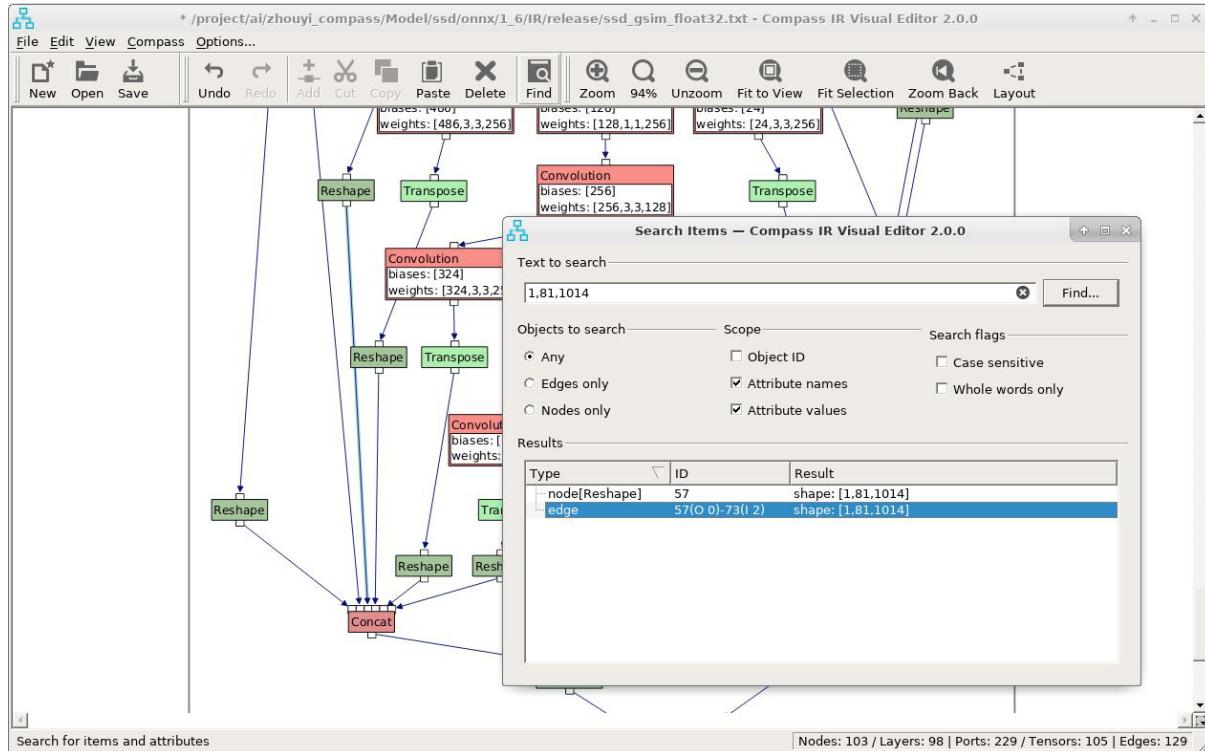
- 增加图的输入:
 1. 新建一个 Node。
 2. 设置 layer_type = Input。
 3. 添加 output port。
 4. 编辑 Tensor 的 input_tensor_index。
- 增加图的输出:
 1. 选择一个 Node。
 2. 鼠标右键输出节点。
 3. 选择 Set graph output。
 4. 编辑 Tensor 的 output_tensor_index。
- 删除图的输入: 删除 Tensor 所在 Input Layer。
- 删除图的输出并删除节点: 删除 Tensor 所在 Layer。
- 删除图的输出但保留节点: 删除辅助节点, 并删除 output_layer_index 字段。
- 修改输入输出顺序: 修改 output_layer_index 字段。

如果输入输出的索引填写错误, 例如填写了相同的 Index, 会在保存时触发自动检查机制, 并告知用户。

查找

在菜单栏单击 Edit > Find, 单击导航栏的 Find 图标或者使用快捷键 Ctrl + F, 可以打开搜索栏查找对象。

图 4-47: 查找

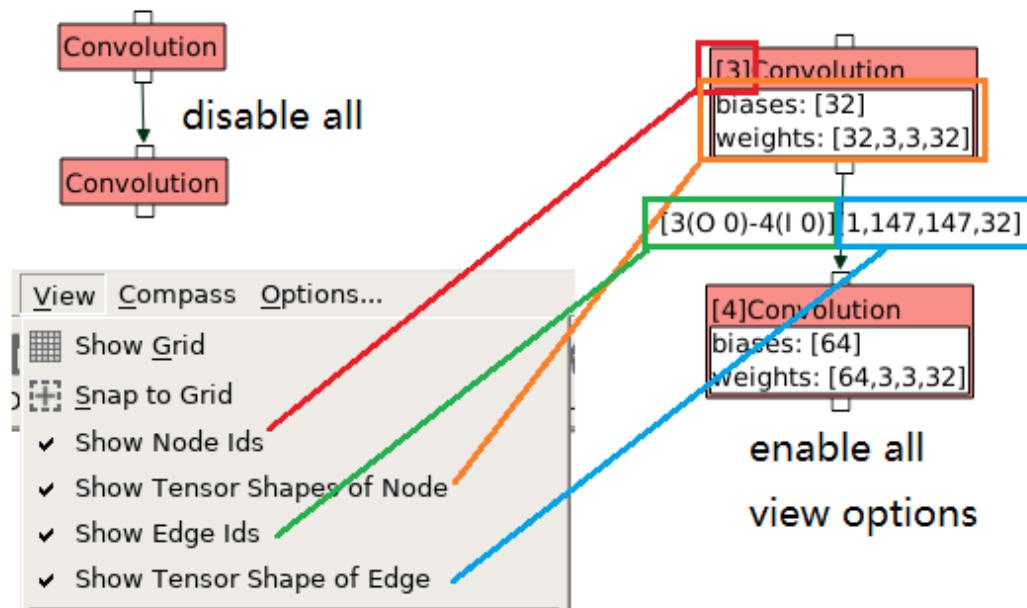


在搜索框中键入关键字即可对 IR 文本内容进行查找。修改 Objects to search 选项可以限定查找对象，修改 Scope 选项可以限定查找键名或者键值。Results 栏中会列出匹配的项目。单击匹配项目，可以把所选项目载入选区，并且视图会自动调整至可视范围内。

修改外观与布局

通过 View 菜单栏可以对当前视图执行放大缩小等操作，也可以改变绘制信息。可配置的选项如图 4-48 所示：文件加载后默认使用 dot 布局。

图 4-48: 外观选项



自动布局后可以通过鼠标左键按住节点拖动来进行手动布局。打乱布局后随时可以使用 **View > Layout** 重新布局。布局使用第三方布局引擎的 Graphviz/dot。你可以在设置页面配置 dot 工具的地址。

自动检查

保存文件会触发两次测试检查图的合理性。第一次检查发生在二位图像保存到 IR 前，检查序列化的可行性。第二次检查会对 IR 执行一次 aipu_checker。测试也可以通过单击 **Menu > Compass > Check (built-in / aipu_checker)** 手动触发。

第一次检查的测试内容包括：

- 存在临时节点
- 存在重名，重 ID 的 layer_name, tensor_name
- 存在未连接的端口
- 存在孤儿节点
- 存在多个 Tensor 接入同一个 Input Port
- 图的输入输出 Tensor 索引不是从 0 开始的递增序列

如果未能通过测试，IR 文件不会被保存，并且会弹出错误信息提示用户修改图内容，直到所有错误被修复。

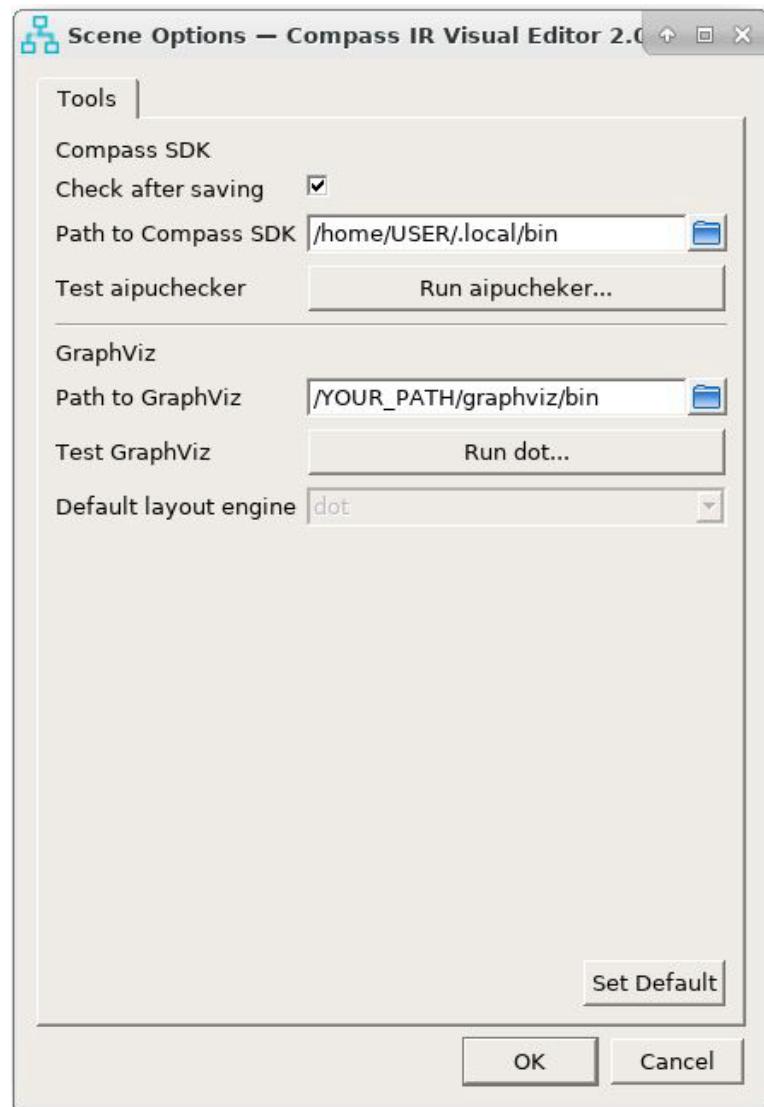
位于 Compass SDK 内的 aipu_checker 会更全面地检查你的 IR 是否符合规范，例如算子 shape 是否超出可支持的规格。关于 aipu_checker 的完整说明，请参考《 Zhouyi Compass NN Compiler User Guide 》。

如果您正在编写自定义算子，在还没实现自定义算子的 aipu_checker 插件前，为了防止误判，可以在设置页面关掉 aipu_checker 的自动触发机制。等写完插件后再开启自动测试。

设置

单击 **Menu > Options...** 可以设置 aipusdk 和 dot 的安装目录并测试工具的可用性。默认情况下，Compass SDK 会安装在 “\$HOME/.local/bin” 。单击 **Set Default** 会重置这一选项。

图 4-49：设置

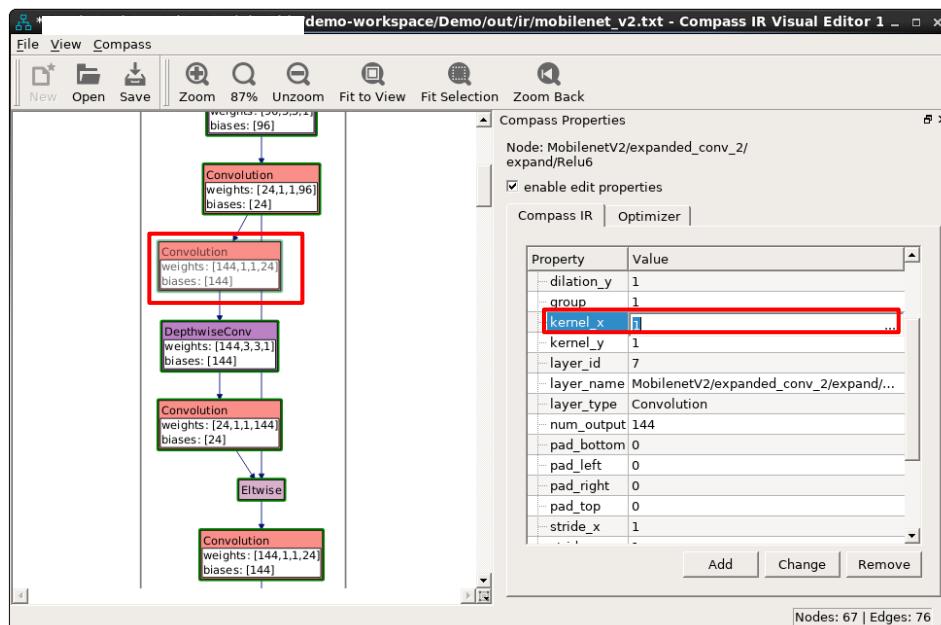


示例 1：修改卷积的 kernel

操作步骤（如图 4-50 所示）

1. 选择要编辑的 Convolution 节点。
2. 在 Compass Properties 区域选择 Compass IR 页签。
3. 单击 kernel_x 的 Value。
4. 输入新值。
5. 保存以生效。

图 4-50：编辑一个节点的属性



示例 2：修改卷积的输出量化系数

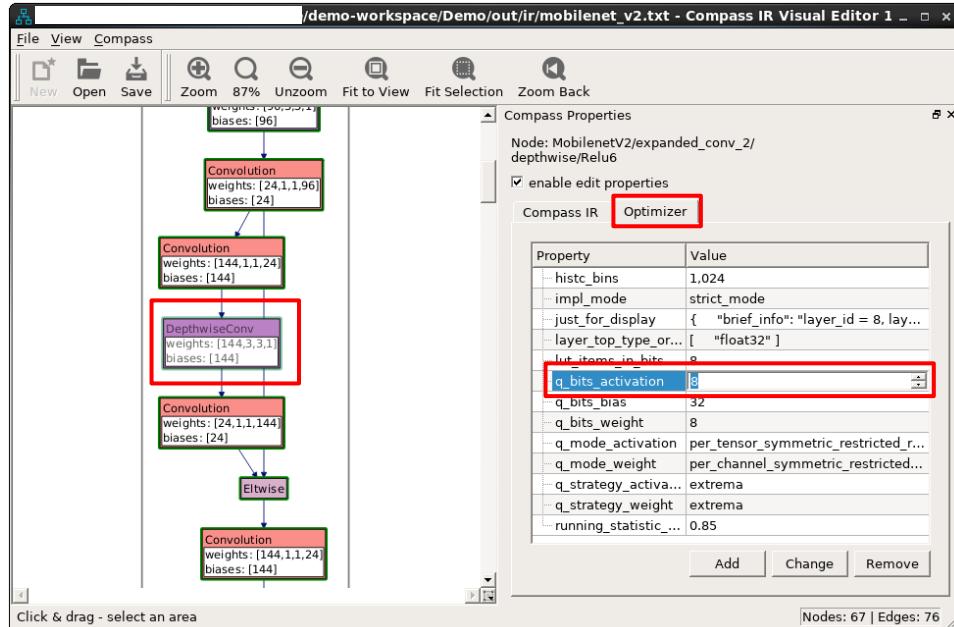
操作步骤（如图 4-51 所示）

1. 选择要编辑的 Convolution 节点。
2. 选择 Optimizer 页签。
3. 选择 q_bit_activation 的 Value。
4. 输入新值。
5. 保存以生效。



只有 Float IR 可以配置量化参数，当 IR 的量化参数可配置时，节点有蓝色边框作为提示。

图 4-51: 编辑一个节点的量化参数

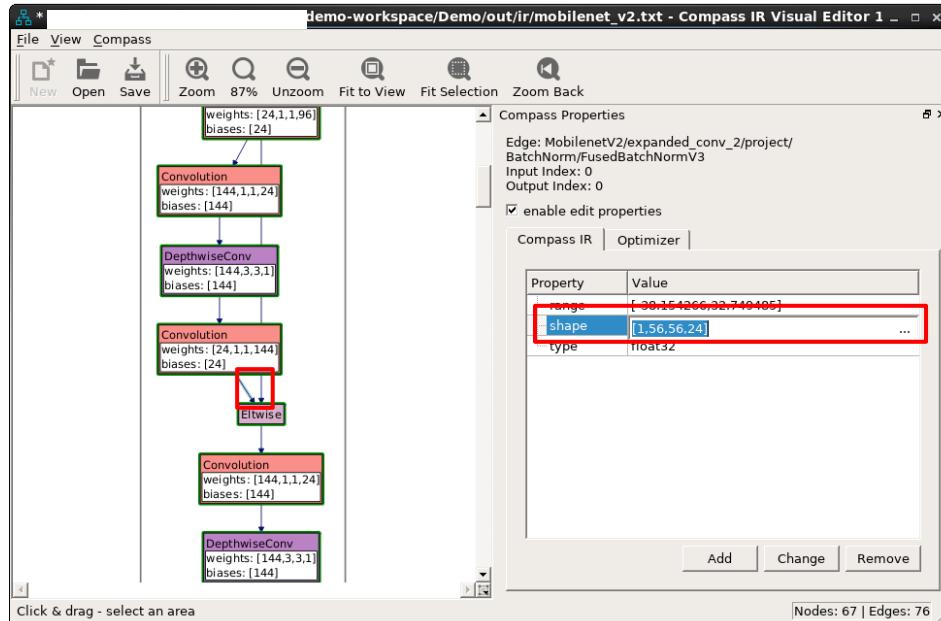


示例 3：修改输出大小

操作步骤（如图 4-52 所示）

1. 选择要编辑的 Convolution 节点和 Eltwise 节点之间的边。
2. 在 Compass Properties 区域选择 Compass IR 页签。
3. 找到 shape 的 Value。
4. 输入新值。
5. 保存以生效。

图 4-52: 编辑一个边的属性



5 算子工程的使用

本章节主要介绍 Compass C/OpenCL 算子工程的创建方式，以及 Compass C/OpenCL 算子工程编译参数的配置。Compass 算子可以使用 Compass C ToolChain 和 Compass OpenCL ToolChain，分别对应 Compass Zhouyi Z2/Z3/X1 和 Compass Zhouyi X2 系列。

5.1 创建算子工程

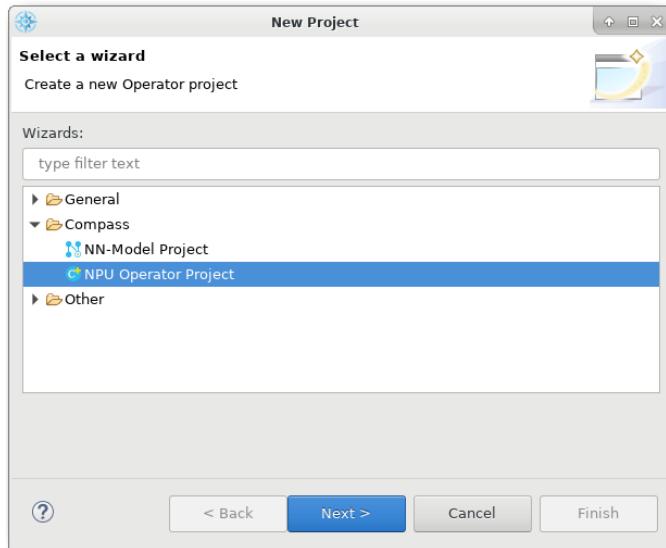
5.1.1 使用模板自动创建项目

创建 Compass C 算子工程

使用模板自动创建 Compass C 算子工程的步骤如下：

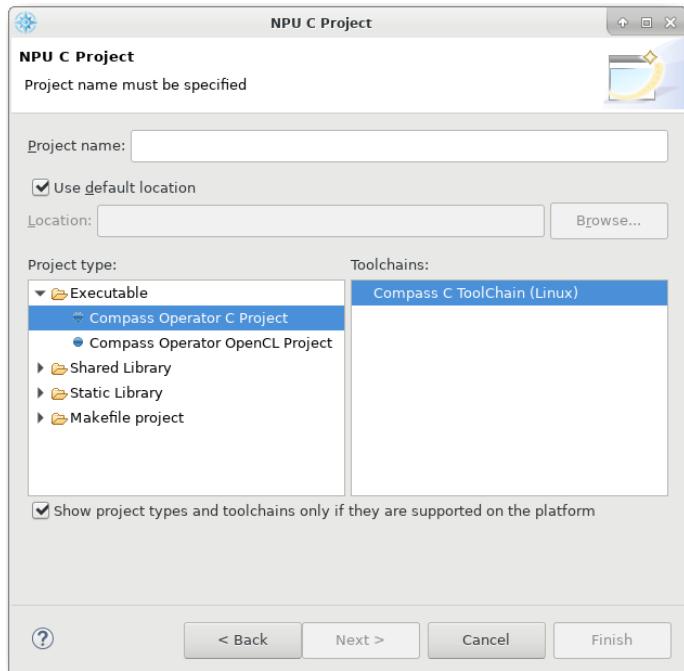
- 在主界面上，单击 **File > New > Project** 创建新的算子工程，如图 5-1 所示。

图 5-1：创建 Compass C 算子工程



- 在弹出的窗口选择工程类型，然后单击 **Next**，进入算子工程向导界面，如图 5-2 所示。

图 5-2: Compass C 算子工程向导界面



3. 在 Project name 文本框中，输入工程名称，然后单击 Next。

算子工程向导界面左边的 Project type 区域会自动选择“Compass Operator C Project”，右边的 Toolchains 区域会显示当前环境拥有的 Toolchains。如果您已正确配置 Compass Toolchain 环境，则会显示“Compass C ToolChain”，选择该 ToolChain 即可。环境配置，请参考[配置 ToolChain 环境](#)。

4. 设置算子工程的基本属性，如图 5-3 所示。

图 5-3: Compass C 算子工程基本设置

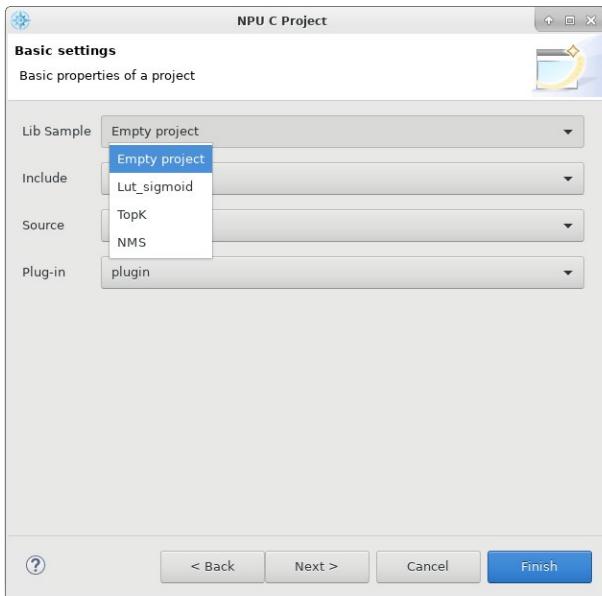
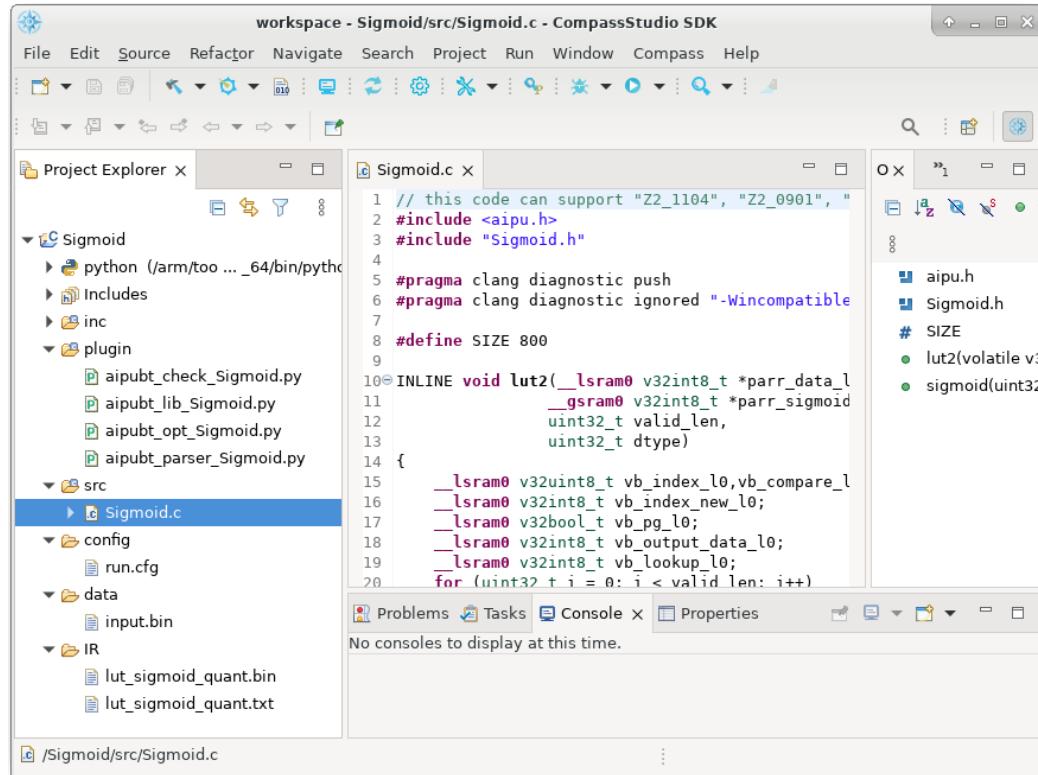


表 5-1: Compass C 算子工程基本属性配置

参数	描述
Lib Sample	Compass C 算子模板工程。您需要根据实际情况，选择对应的模板。目前 CompassStudio 针对算子工程提供五个模板：Empty project、Lut_sigmoid、TopK、NMS、Decodebox。
Include	自定义头文件的目录，默认不修改。CompassStudio 会将该文件自动加入到环境中。如果您有自定义的头文件，则应该放在该目录下。
Source	源码文件，默认不修改。自定义的源码文件应该放在该目录下，并且需要保持源码文件和工程名一致。
Plug-in	算子 plugin 脚本，默认不修改。CompassStudio 对该目录支持 Python 语法的解析及运行。编写算子 plugin 的脚本应该放在该目录下。 如果您需要自定义 Python 环境的目录，请参考 8.2 算子工程中增加 Python 语法 。

5. 查看工程结构。以 Sigmoid 模板为例，算子工程创建后的结构如图 5-4 所示。

图 5-4: Compass C 算子工程结构



Compass C 算子工程包含 Include、inc、plugin、src、config、data 和 IR 目录，如表 5-2 所示。

表 5-2: Compass C 算子工程目录结构

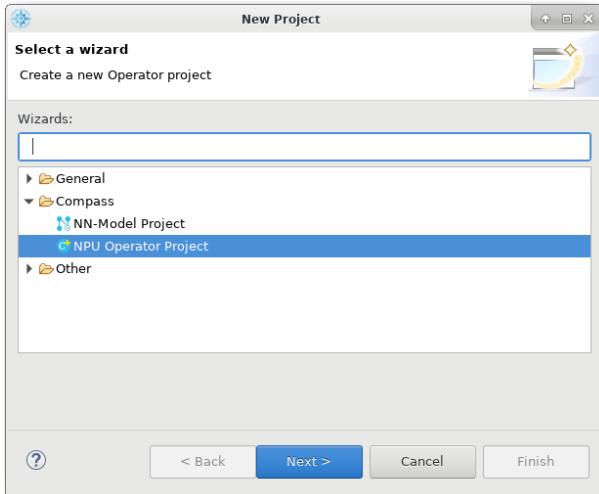
目录	描述
Include	引用配置 ToolChain 环境变量的库文件。
inc	自定义头文件的目录。
plugin	算子 plugin 脚本。
src	Compass C 源码文件。
config	算子调试需要的配置文件。
data	算子调试需要的输入数据。
IR	算子调试需要的 IR 文件。

创建 Compass OpenCL 算子工程

使用模板自动创建 Compass OpenCL 算子工程的步骤如下：

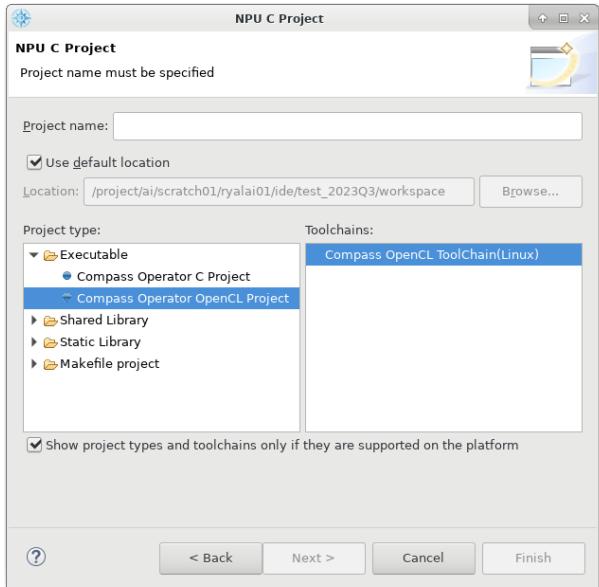
- 在主界面上，单击 File > New > Project 创建新的算子工程，如图 5-5 所示。

图 5-5: 创建 Compass OpenCL 算子工程



- 在弹出的窗口选择工程类型，然后单击 **Next**，进入算子工程向导界面，如图 5-6 所示。

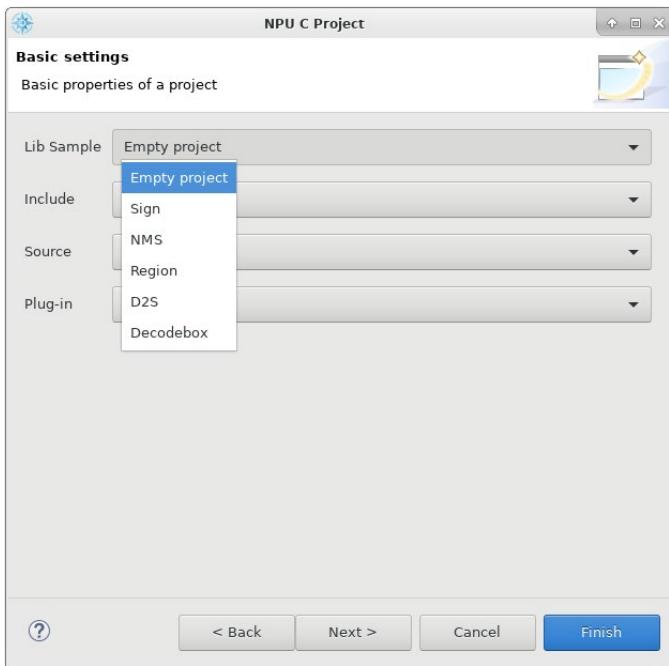
图 5-6: Compass OpenCL 算子工程向导界面



- 在 Project name 文本框中，输入工程名称，然后单击 **Next**。

算子工程向导界面左边的 Project type 区域选择“Compass Operator OpenCL Project”，右边的 Toolchains 区域会显示当前环境拥有的 Toolchains。如果您已正确配置 Compass Toolchain 环境，则会显示“Compass OpenCL ToolChain”，选择该 ToolChain 即可。环境配置，请参考[配置 ToolChain 环境](#)。

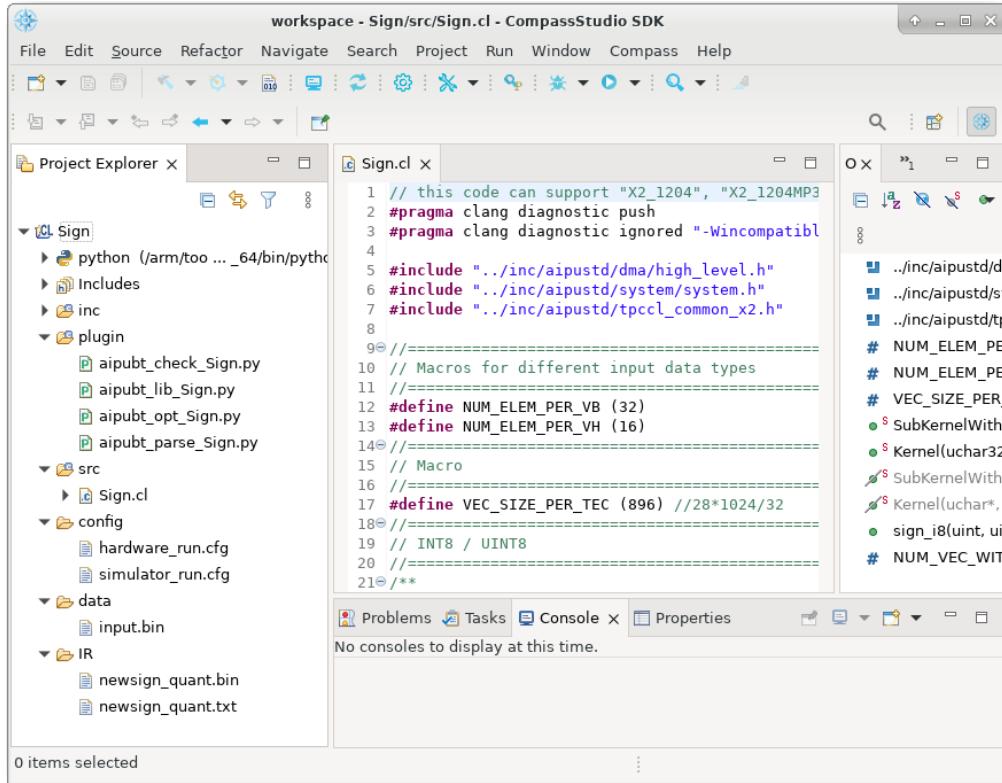
- 设置算子工程的基本属性，如图 5-7 所示。

图 5-7: Compass OpenCL 算子工程基本设置**表 5-3: Compass OpenCL 算子工程基本属性配置**

参数	描述
Lib Sample	Compass OpenCL 算子模板工程。您需要根据实际情况，选择对应的模板。目前 CompassStudio 针对算子工程提供五个模板：Empty project、Sign、NMS、Region、Decodebox。
Include	自定义头文件的目录，默认不修改。CompassStudio 会将该文件自动加入到环境中。如果您有自定义的头文件，则应该放在该目录下。
Source	源码文件，默认不修改。自定义的源码文件应该放在该目录下，并且需要保持源码文件和工程名一致。
Plug-in	算子 plugin 脚本，默认不修改。CompassStudio 对该目录支持 Python 语法的解析及运行。 编写算子 plugin 的脚本应该放在该目录下。 如果您需要自定义 Python 环境的目录，请参考 8.2 算子工程中增加 Python 语法 。

5. 查看工程结构。以 Sign 模板为例，算子工程创建后的结构如图 5-8 所示。

图 5-8: Compass OpenCL 算子工程结构



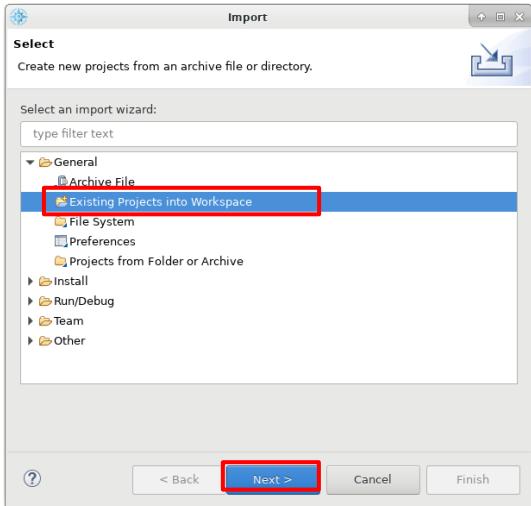
Compass OpenCL 算子工程包含 Include、Inc、plugin、src、config、data 和 IR 目录，如表 5-4 所示。

表 5-4: Compass OpenCL 算子工程目录结构

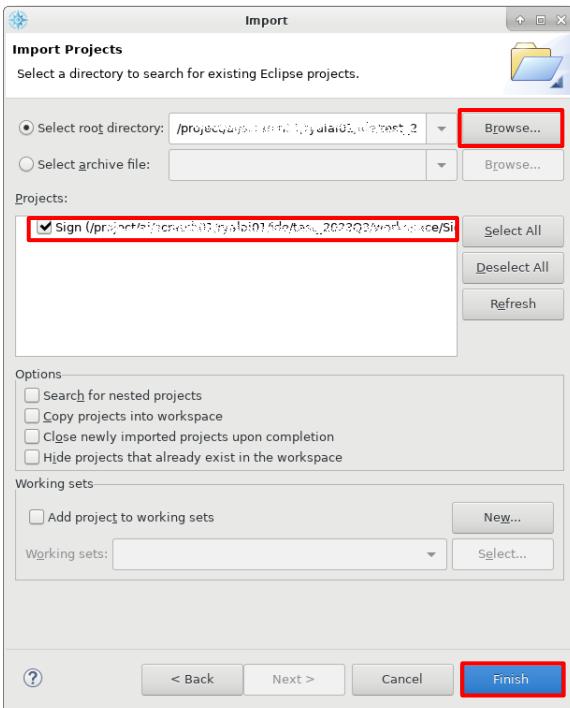
目录	描述
Include	引用配置 ToolChain 环境变量的库文件。
inc	自定义头文件的目录。
plugin	算子 plugin 脚本。
src	Compass OpenCL 源码文件。
config	算子调试需要的配置文件。
data	算子调试需要的输入数据。
IR	算子调试需要的 IR 文件。

5.1.2 从已有项目直接导入创建项目

- 右键单击 CompassStudio 左侧的工程管理列表区，选择 Import。在弹出的界面中选择 General > Existing Projects into Workspace，如图 5-9 所示。

图 5-9: 导入工程

2. 单击 **Next**, 进入下一步界面。
3. 单击 **Browse**, 选择需要导入项目的地址, 在 **Projects** 区域选择需要导入的项目。然后单击 **Finish**, 完成项目的导入, 如图 5-10 所示。

图 5-10: 选择项目地址

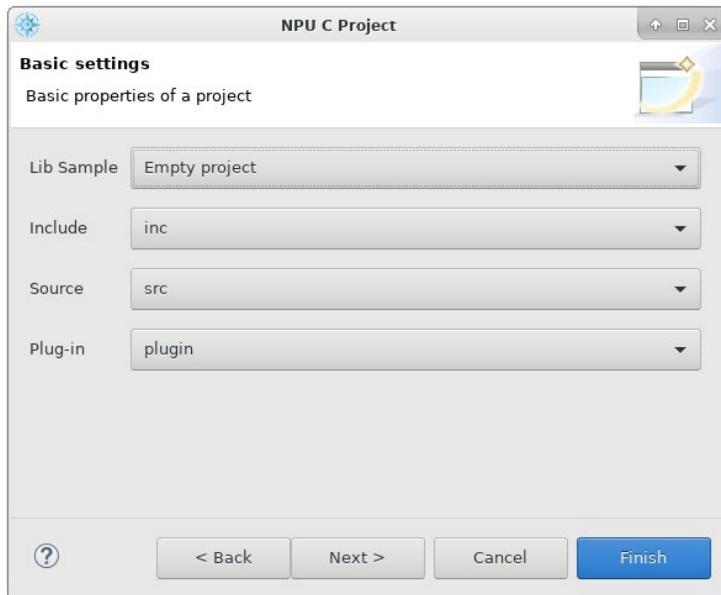
导入的工程只能是 CompassStudio 导出的工程。非 CompassStudio 导出的工程无法正常导入。

5.1.3 手动创建项目

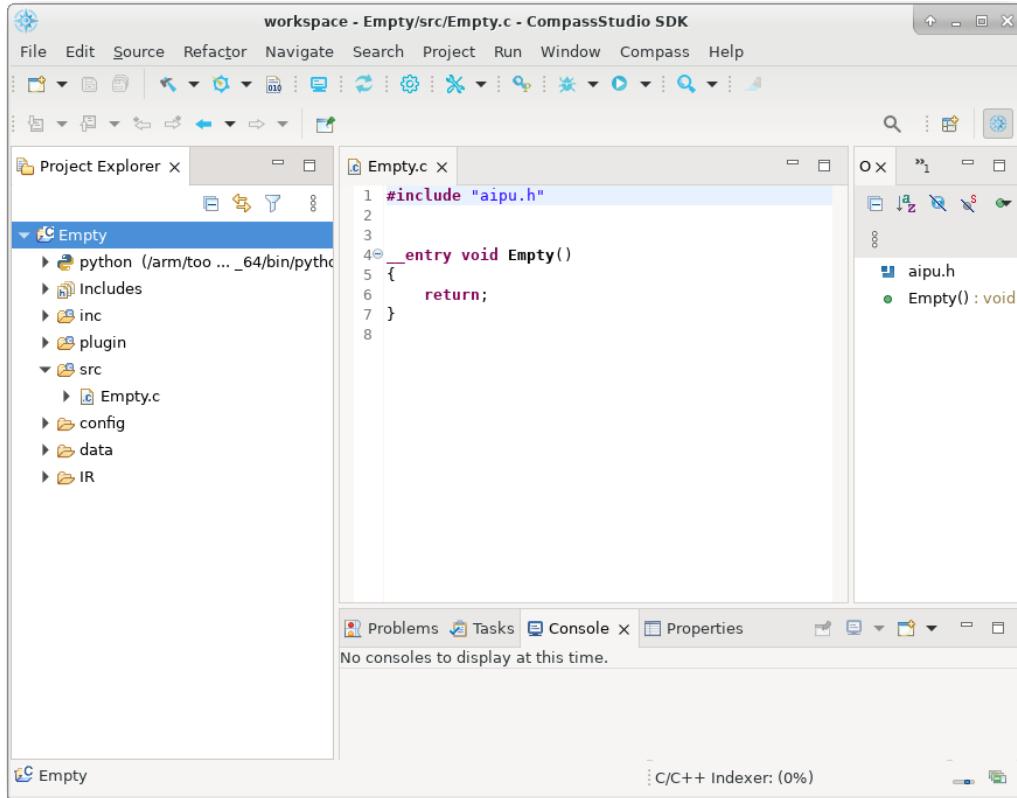
手动创建 Compass C/OpenCL 算子工程的步骤一致。本小节以手动创建 Compass C 算子为例，主要步骤如下：

1. 参考 [5.1.1 使用模板自动创建项目](#)，在算子工程基本设置界面的 Lib Sample 下拉列表框中，选择 Empty project，如图 5-11 所示。

图 5-11：选择模板



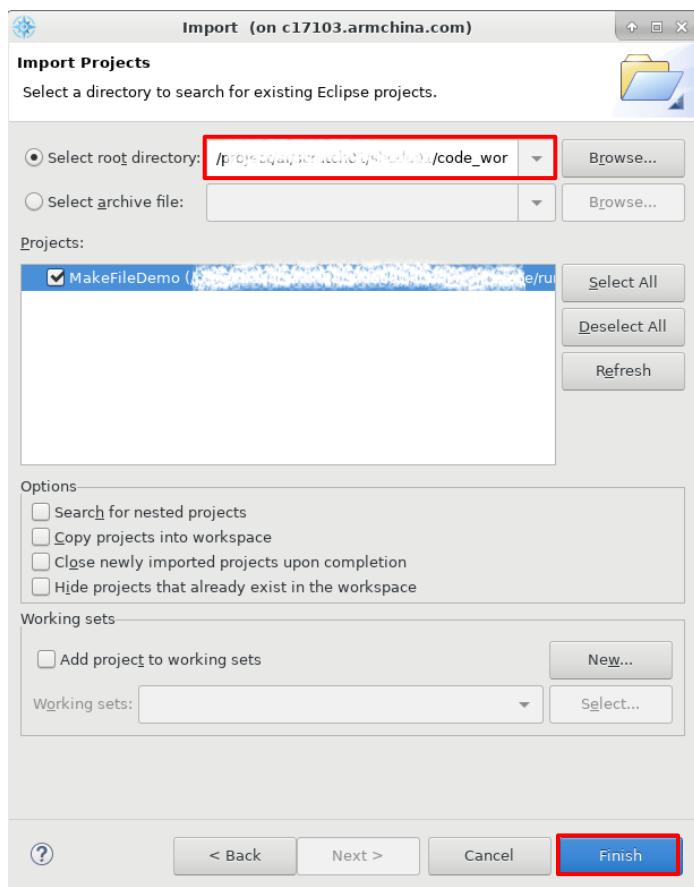
2. 单击 Next 直到项目被创建，如图 5-12 所示。

图 5-12: 手动创建项目

3. 如图 5-12 所示, 手动创建的工程需要您自定义算子 和 plugin。CompassStudio 提供了一个 Empty (void) 的算子 .c 文件, 您可以根据实际需求自行修改:
 - 在 inc 目录下创建自定义的头文件。CompassStudio 已经自动将该目录加入到 include 环境中。
 - 在 plugin 目录下创建自定义的 plugin 脚本。CompassStudio 默认对该目录支持 Python 语法 (关于 plugin 命名, 请参考 Compass OP plugin) 。
 - 在 config 目录下自定义算子调试的配置文件, 并命名为 “run.cfg” 。
 - 在 data 目录下自定义算子的输入数据文件。
 - 在 IR 目录下自定义算子的 IR 文件。

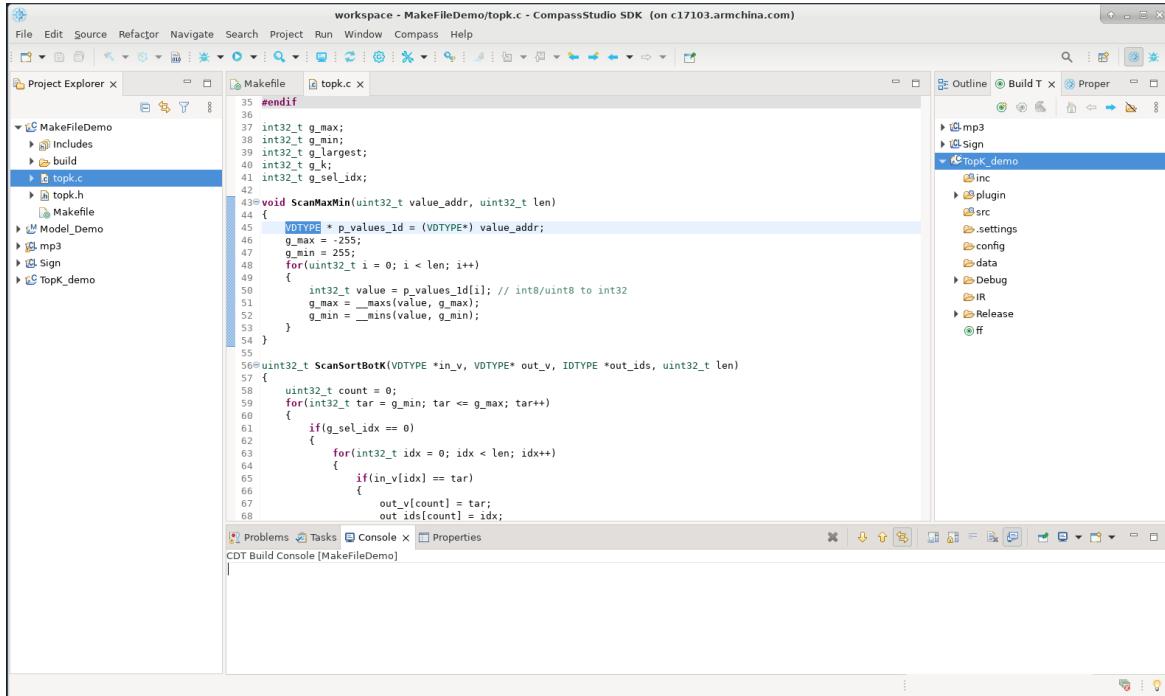
5.1.4 基于已有的 Makefile 创建项目

1. 右键单击已有的 Makefile, 然后选择 New > Import > General > Existing Projects into Workspace, 如图 5-13 所示。

图 5-13: 导入 Makefile 工程

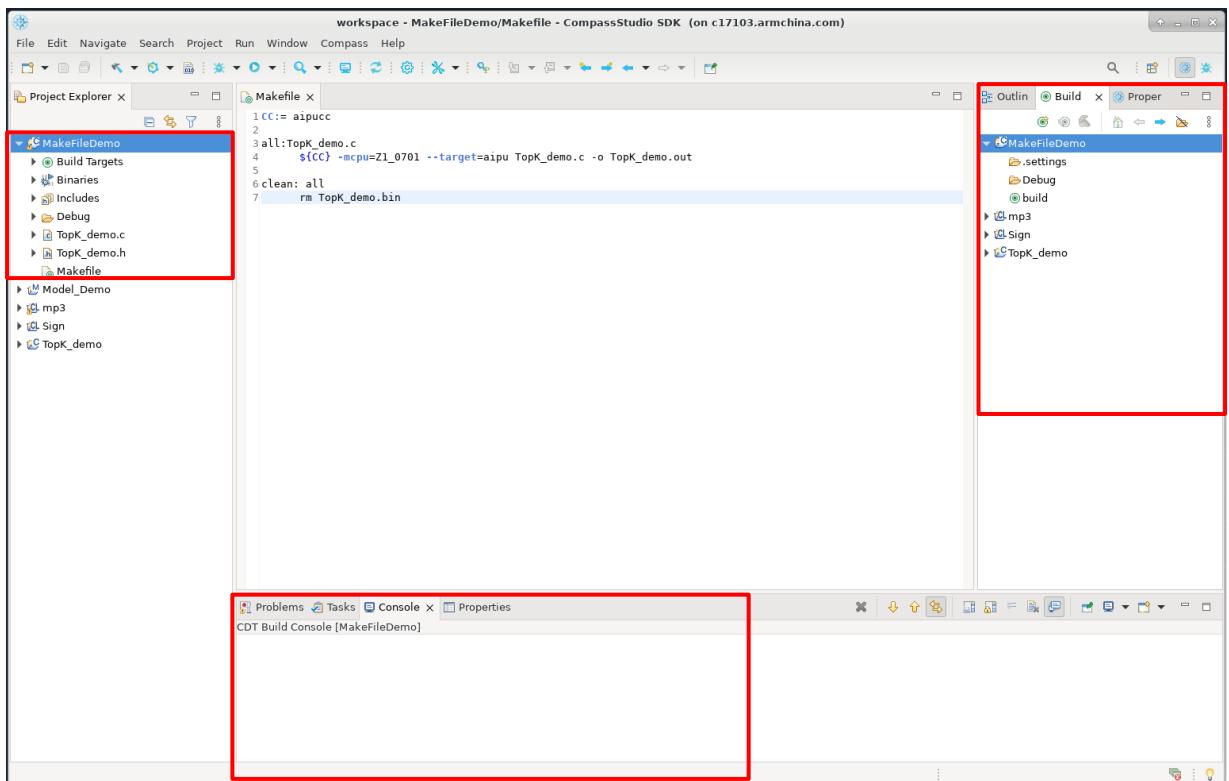
2. 单击 **Finish**, 完成导入, 如图 5-14 所示。

图 5-14: 导入 Makefile 工程



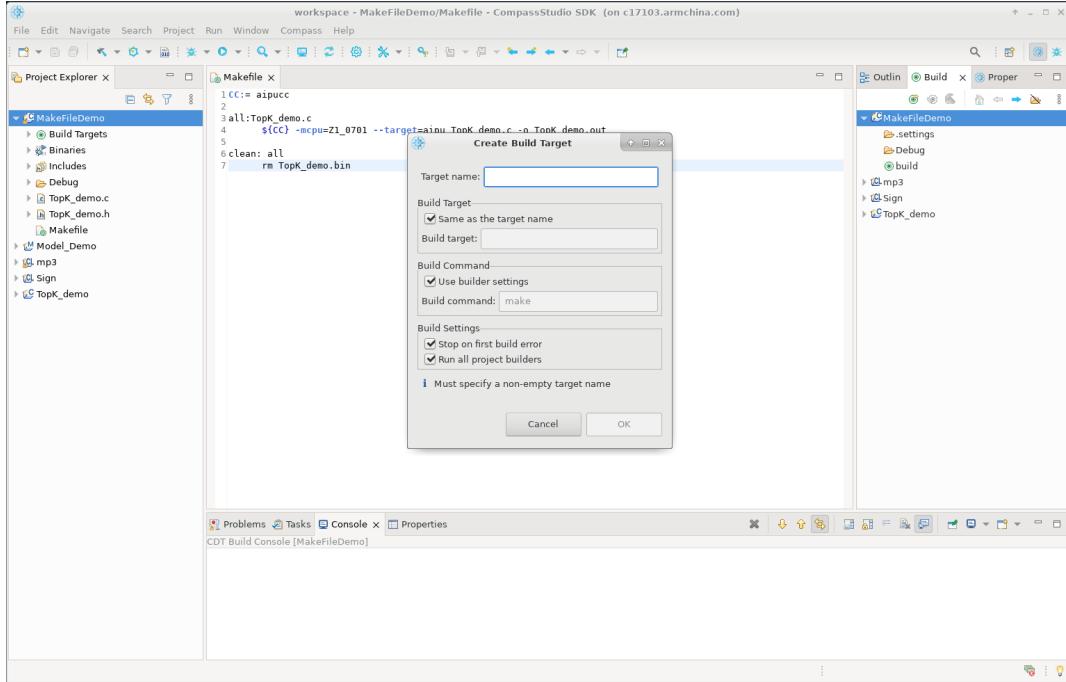
3. 单击 Build 图标，编译过程如图 5-15 所示。

图 5-15: 编译 Makefile 工程



4. 在图 5-15 所示的界面中, 您也可以单击 Window > Show View > Other > Build Targets, 通过 Build Targets 视图管理 Makefile 工程, 如图 5-16 所示。

图 5-16: 管理 Makefile 工程



5. 在图 5-16 所示的 Build Targets 视图中, 右键单击 Makefile 工程, 然后选择 New 来创建 Build Target。双击创建的 Build Target 即可编译。

5.2 配置算子工程编译参数

Compass C/OpenCL 算子编译过程如下:

- 周易 Z2/Z3/X1:

C代码 (.c) > 经过汇编器 (.s) > 经过编译器 (.o) > 经过链接器 (.out) > 经过GBuild (.bin)

- 周易 X2:

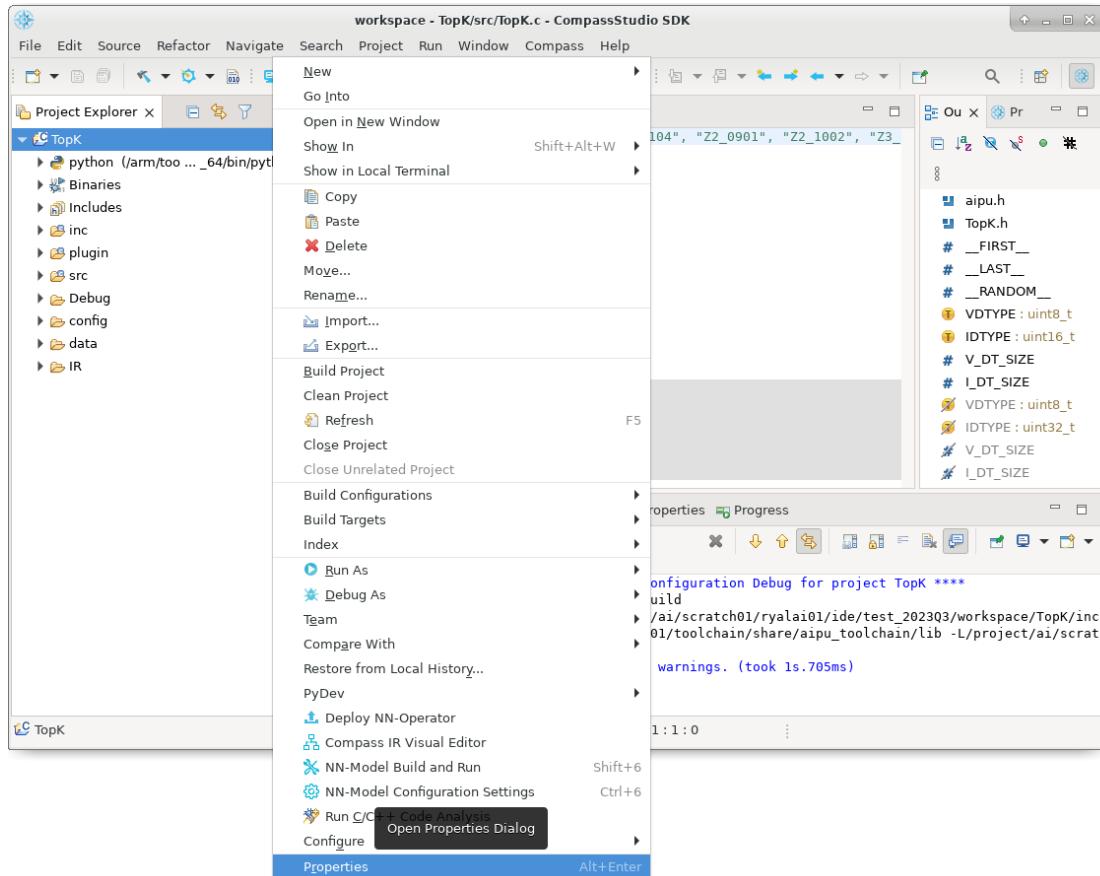
Opencl代码 (.cl) > 经过汇编器 (.s) > 经过编译器 (.o) > 经过 GBuild (.bin)

5.2.1 配置 Compass C 算子工程编译参数

本小节以 TopK 模板为例。请参考 [5.1.1 使用模板自动创建项目](#)来创建 TopK 的算子工程。

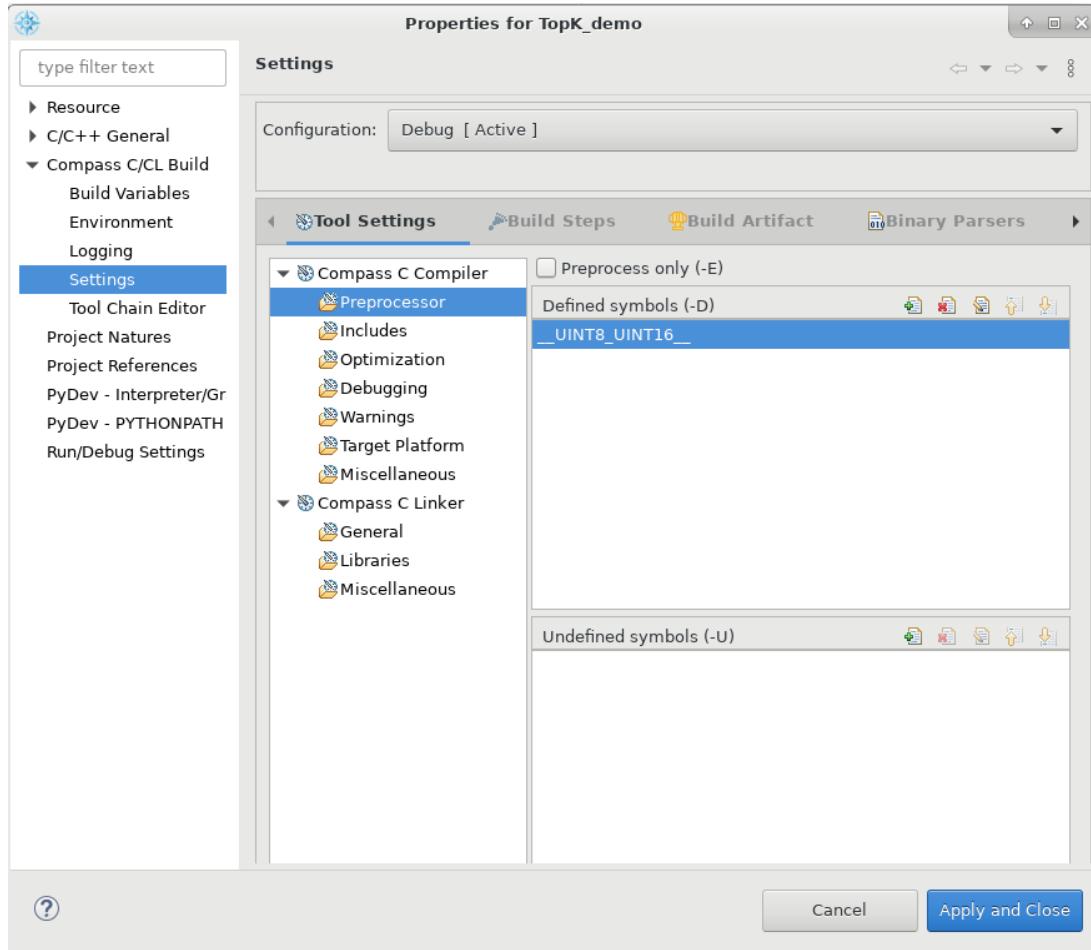
工程创建完成后，右键单击该工程，然后选择 **Properties**，如图 5-17 所示。

图 5-17: 算子工程属性



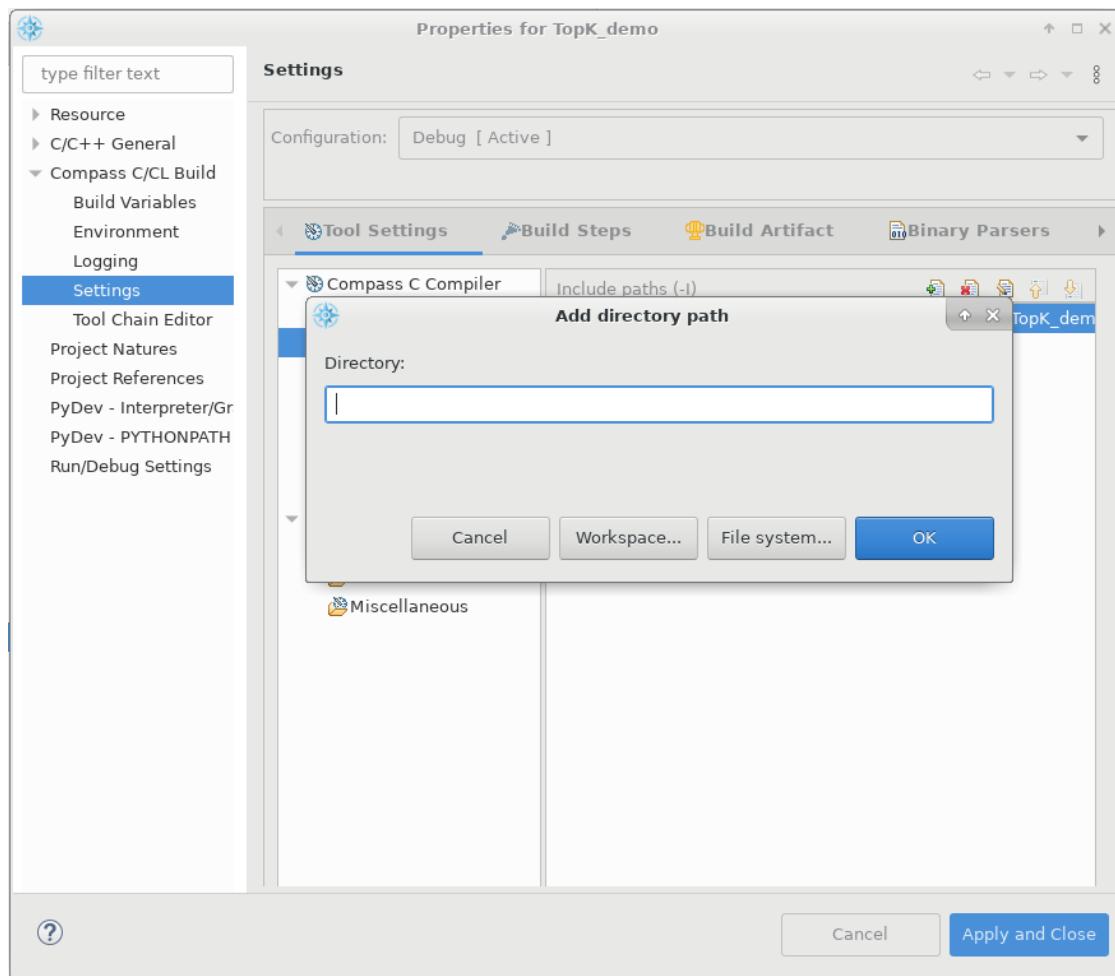
配置 Compiler 参数

- 在工程 Properties 界面，选择 **Compass C/CL Build > Settings > Tool Settings > Compass C Compiler**，如图 5-18 所示。

图 5-18: 算子工程 Compiler 参数配置

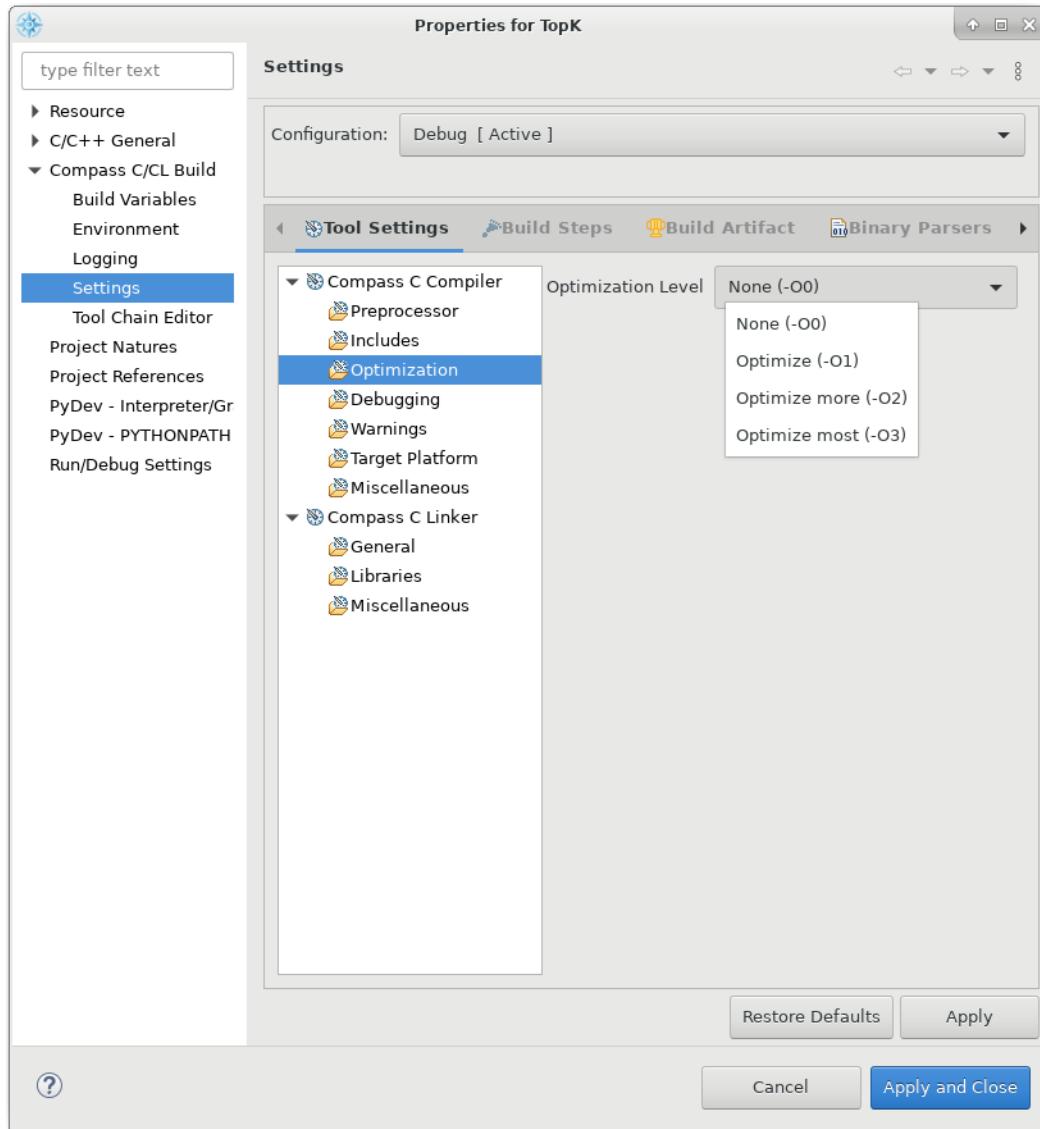
2. 在 **Preprocessor** 中可进行宏定义。以该 TopK 工程为例，源码中定义了多个宏，您需要根据需求，在编译时进行宏选择。单击 进行宏添加（需在源码中定义）。
3. 在 **Includes** 中进行头文件引用。CompassStudio 默认将工程的“inc”目录作为自定义的头文件引用目录。您也可以自定义目录，如图 5-19 所示。

图 5-19: 配置 Compiler 自定义的 include 目录



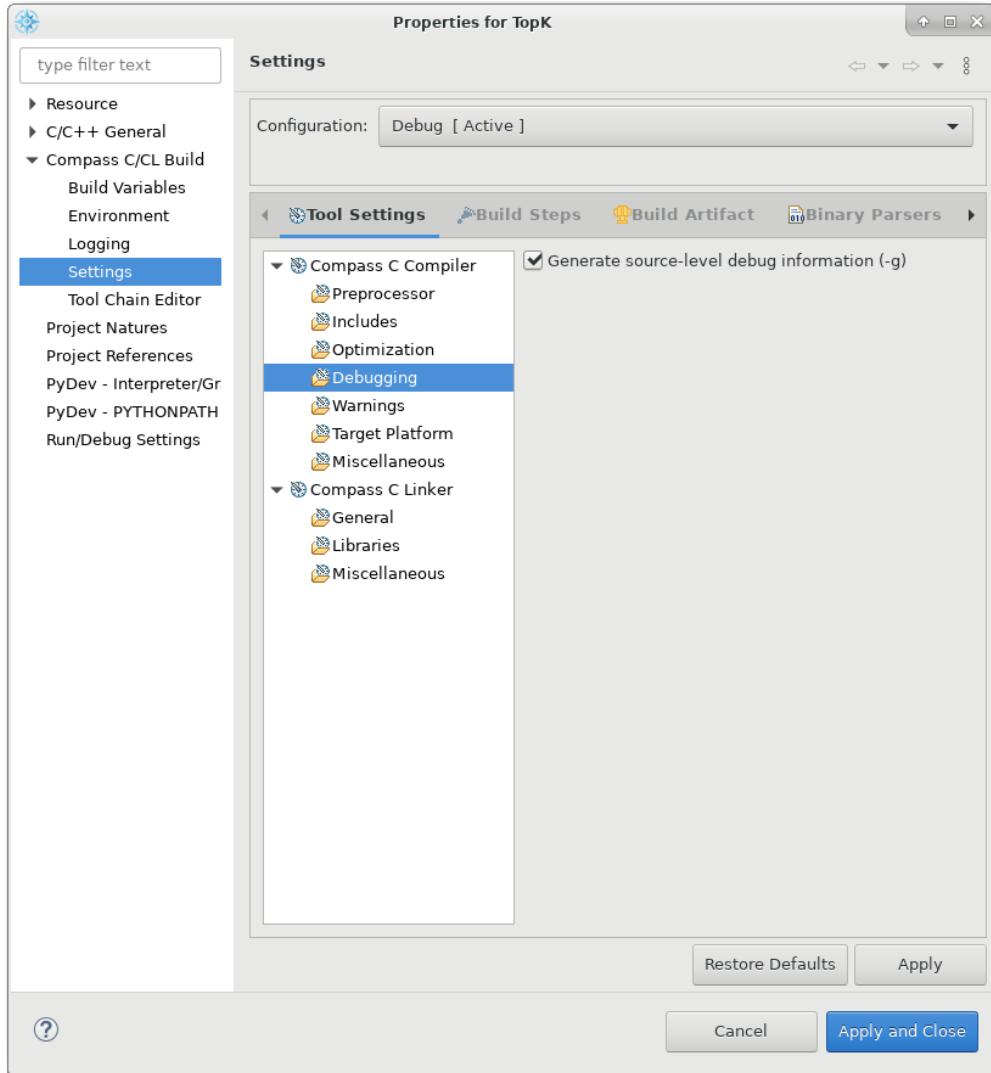
4. 在 Optimization 中可选择 Optimization Level，如图 5-20 所示。

图 5-20: 配置 Compiler 的 Optimization Level



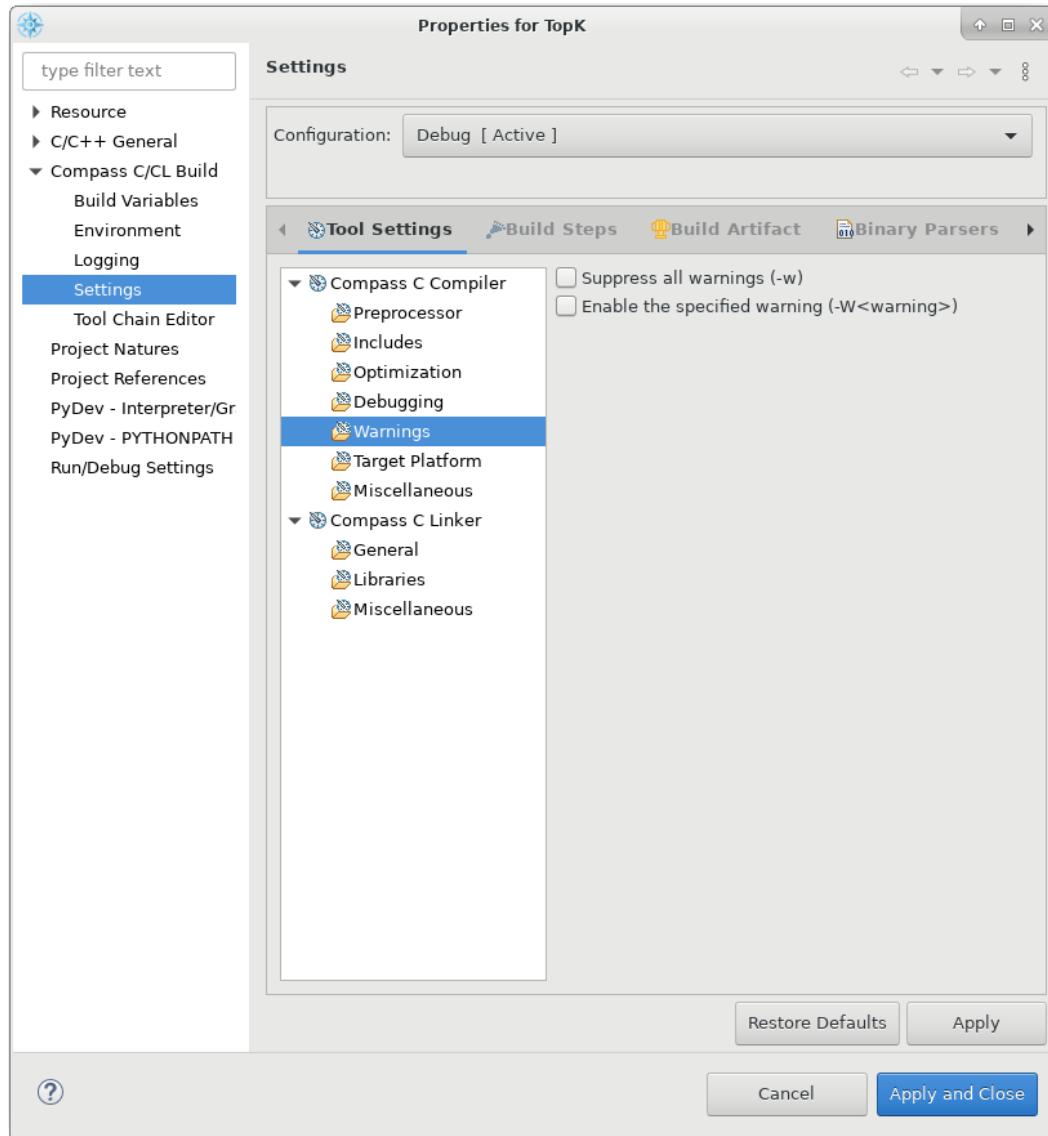
5. 在 Debugging 中选择调试参数，如图 5-21 所示。

图 5-21: 配置 Compiler 的 Debugging 等级

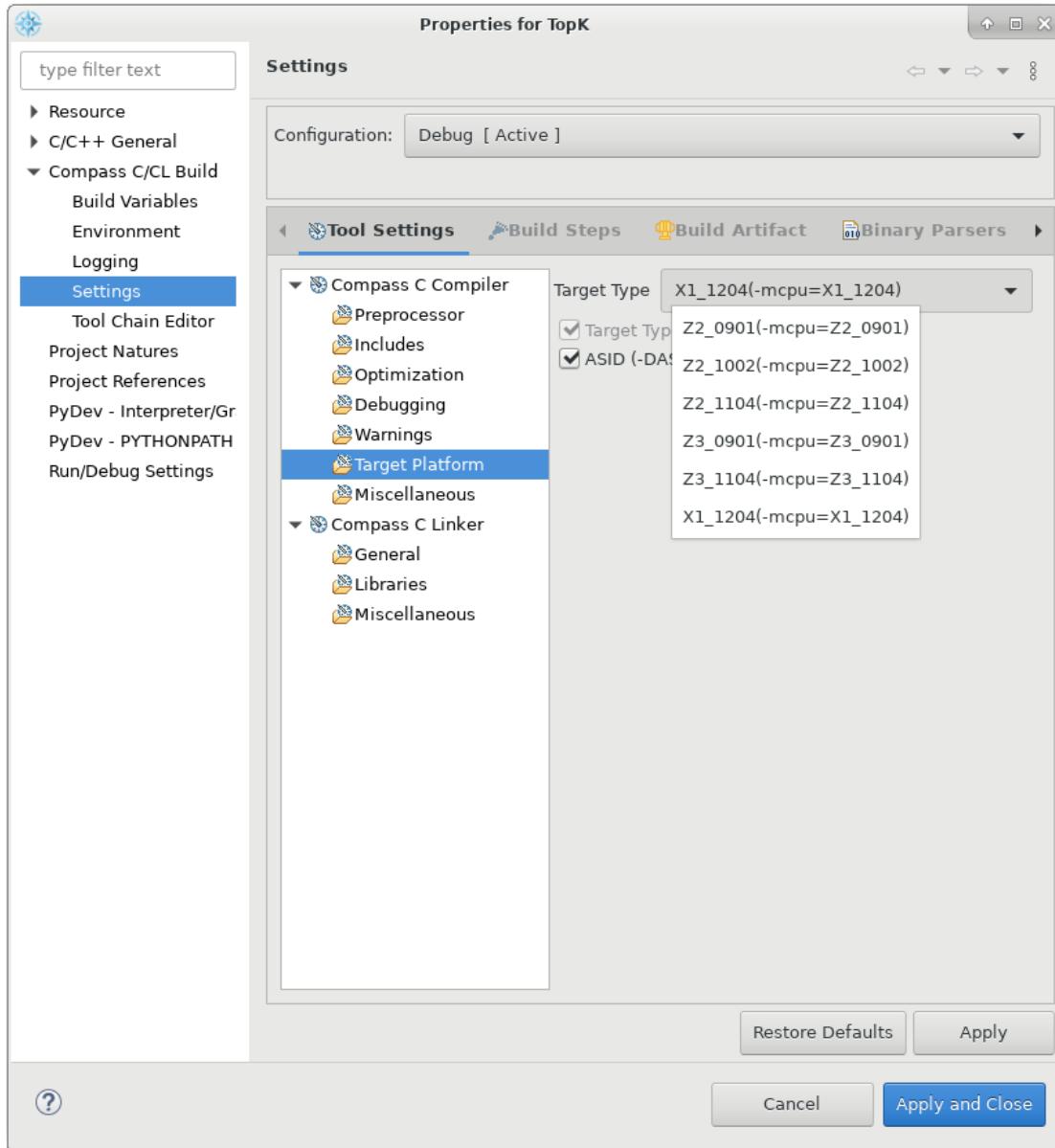


6. 在 Warnings 中可选择 warning 参数，如图 5-22 所示。

图 5-22: 配置 Compiler 的 Warning 参数

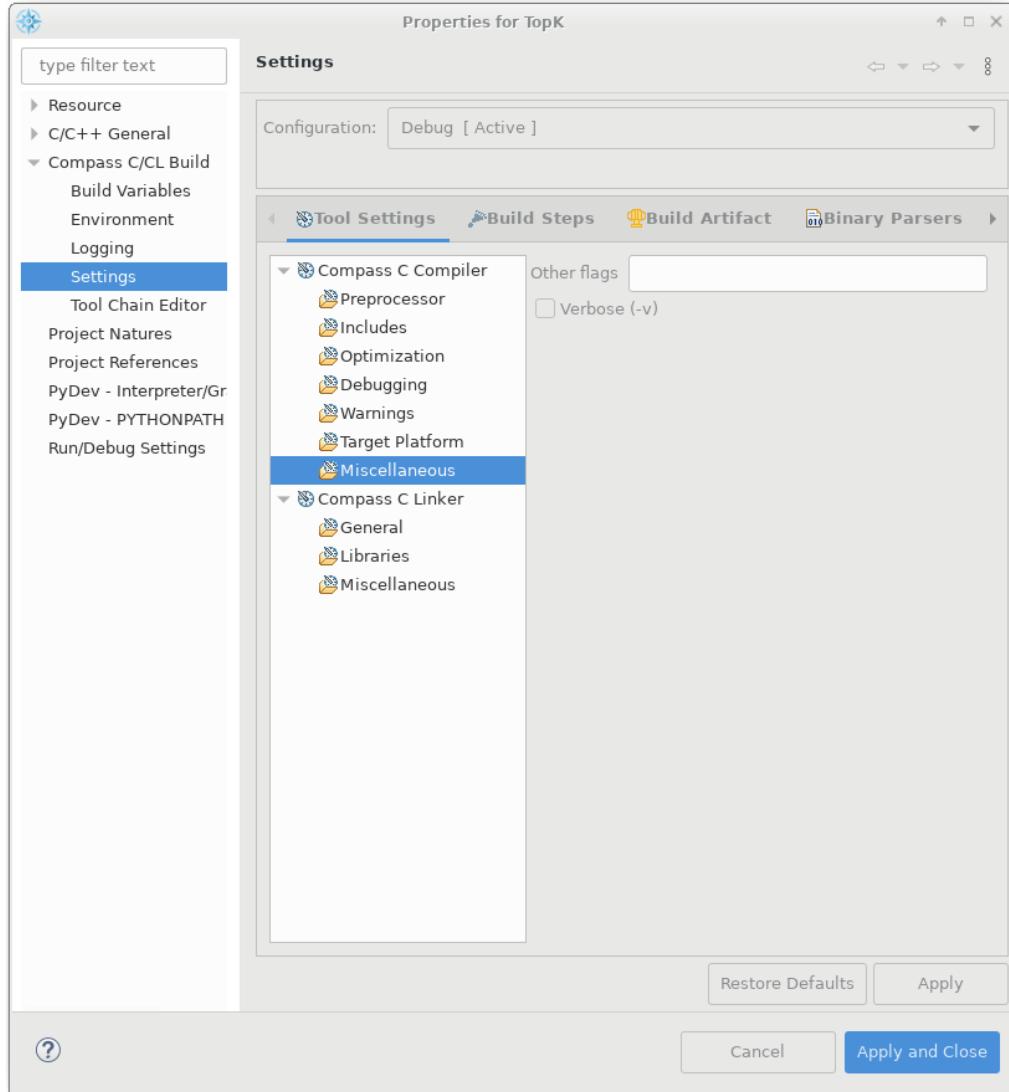


7. 在 Target Platform 中可选择 Target Type 等参数, 如图 5-23 所示

图 5-23: 配置 Compiler 的 Target

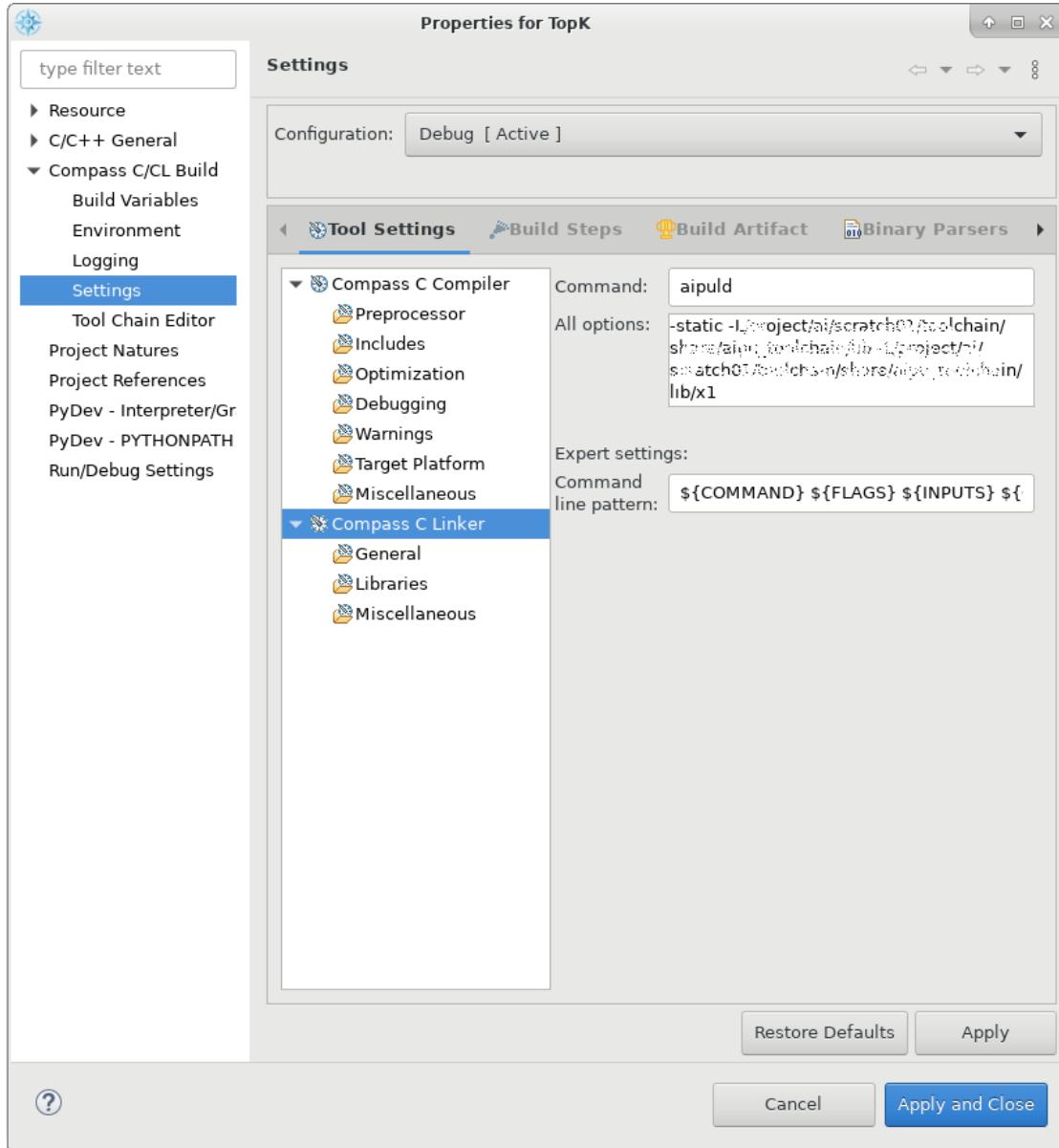
8. 在 Miscellaneous 中可选择其他编译参数，如图 5-24 所示。

图 5-24: 配置 Compiler 的其他编译参数

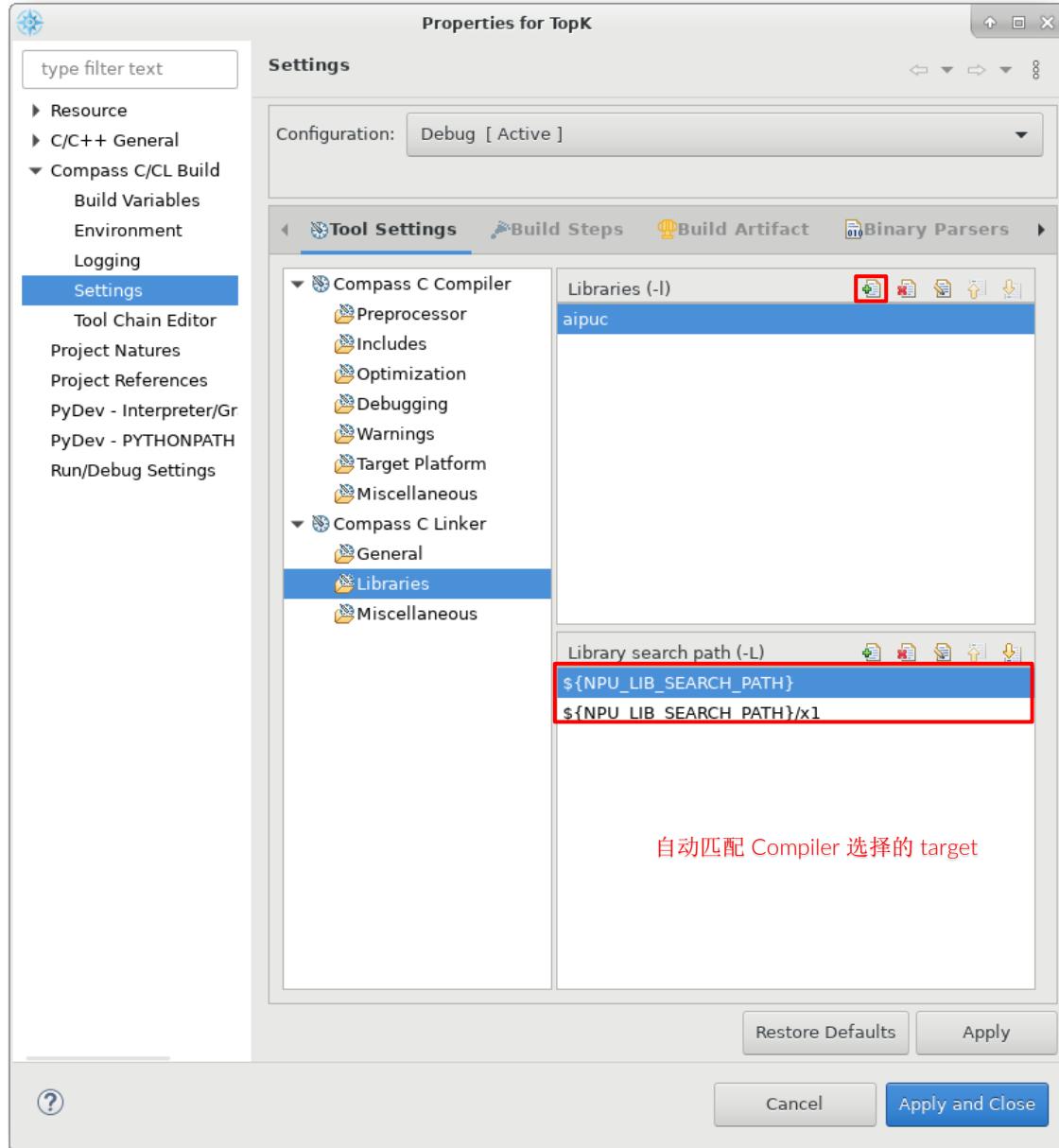


配置 Linker 参数

1. 在工程 Properties 界面, 选择 Compass C Build > Settings > Tool Settings > Compass C linker , 如图 5-25 所示。

图 5-25: 算子工程 Linker 参数配置

2. 在 **Libraries** 中可选择 linker 依赖的 Libraries path，如图 5-26 所示。

图 5-26: 配置 Linker 依赖的 Libraries

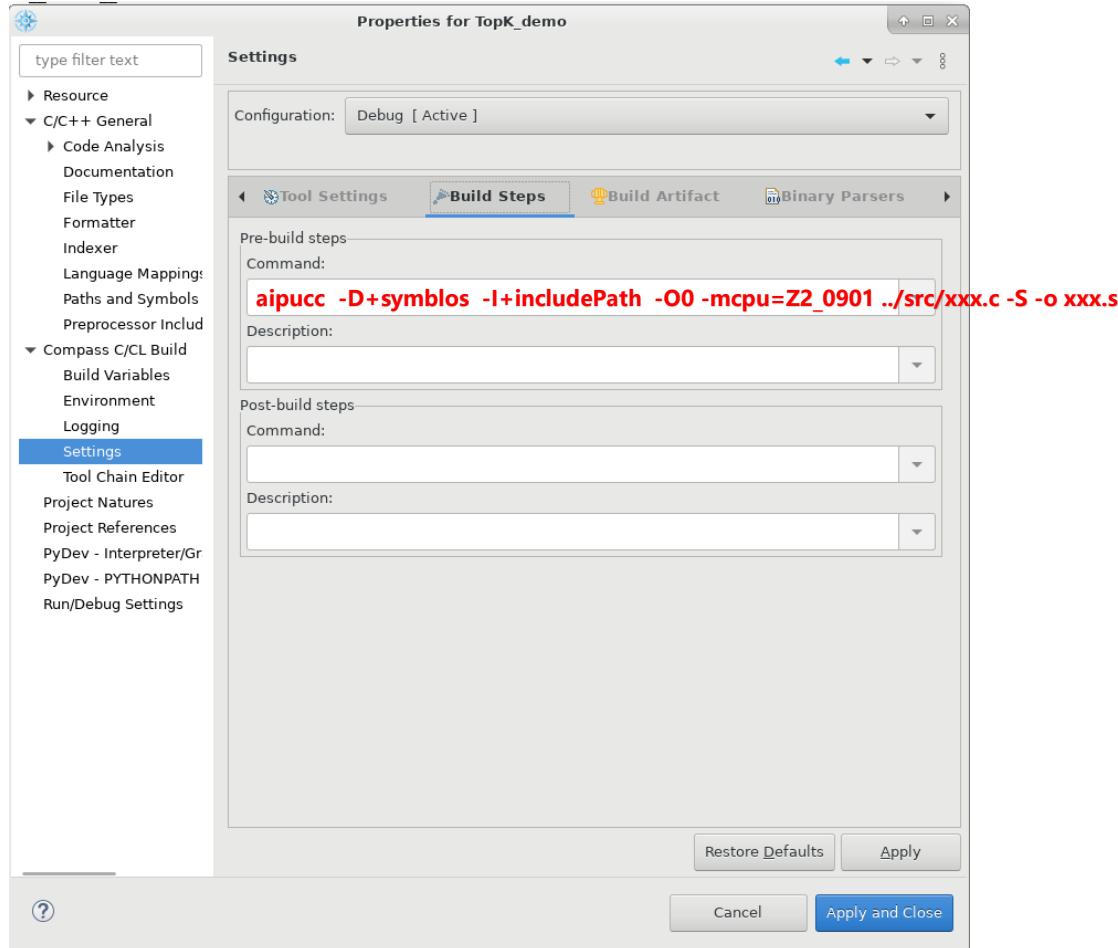
3. 单击 可添加 linker 需要的 lib 库地址。CompassStudio 会根据 [5.2.1 配置 Compiler 参数](#)中选择的 target，自动匹配对应 target 需要的 library。
4. 在 **Miscellaneous** 中可选择其他链接参数。

配置 Assembler 参数

目前 CompassStudio 不支持 Assembler 的参数选择，即用户无需配置 Compass Assembler。想要生成汇编文件，可以参考如下操作：

1. 右键单击工程，打开工程 Properties 页面，然后选择 **Compass C Build > Settings > Build Steps**，如图 5-27 所示。

图 5-27: Assembler 配置页面



2. 可以选择在 Pre-build steps (编译前生成汇编) 或者 Post-build steps (编译完成后生成汇编) 中输入：

```
aipucc -D+symbols -l+includePath -O0 -mcpu= target.xxx xxx.c -S -o xxx.s
```

3. 按照该配置，单击 **Apply and Close**，即可对 Assembler 进行配置。请参考 [6.1.3 编译生成汇编文件](#)。

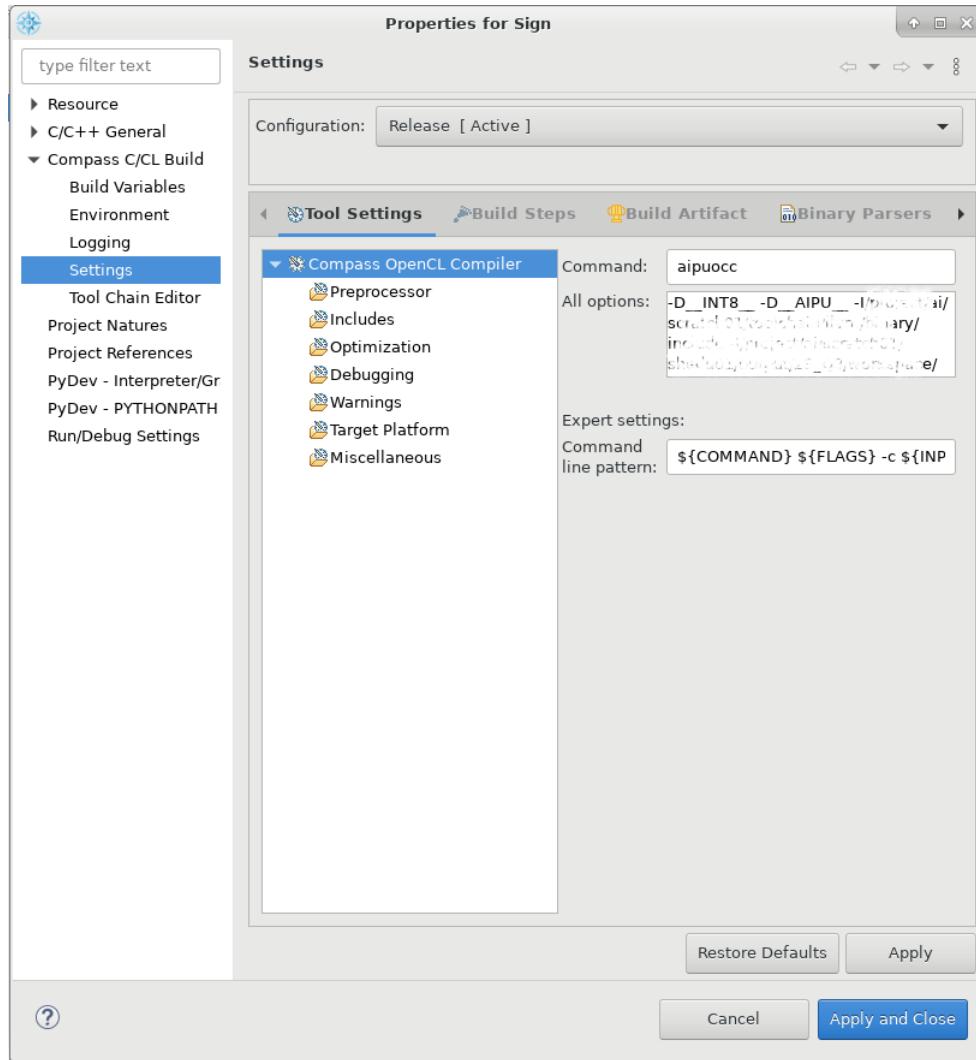
5.2.2 配置 Compass OpenCL 算子工程编译参数

Compass OpenCL 算子工程只需要配置 Compiler 参数即可，如需要生成并查看汇编代码，请参考[配置 Assembler 参数](#)。

配置 Compiler 参数

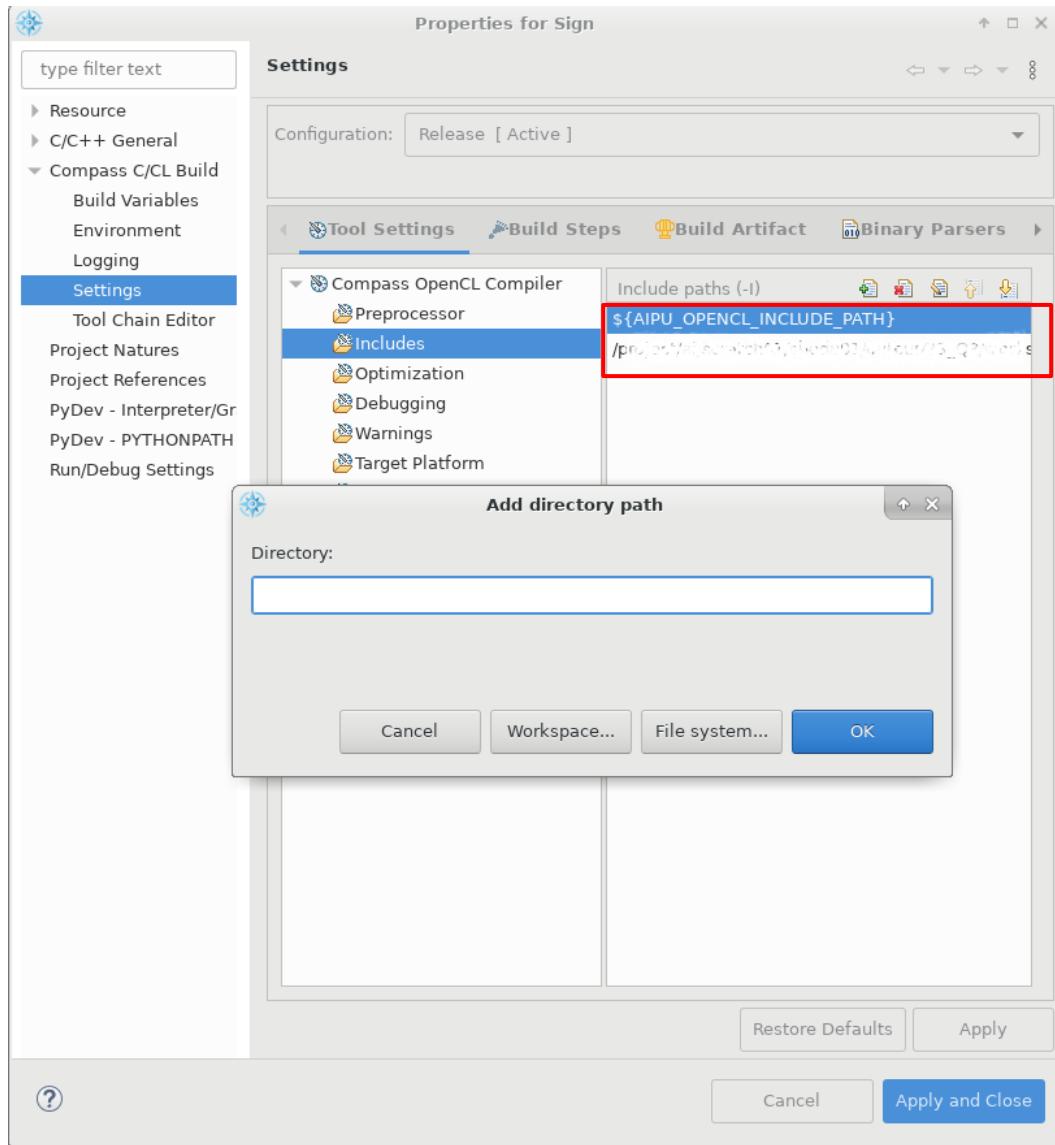
- 在工程 Properties 界面, 选择 **Compass C/CL Build > Settings > Tool Settings > Compass OpenCL Compiler**, 如图 5-28 所示。

图 5-28: 算子工程 Compiler 参数配置



- 在 **Preprocessor** 中可进行宏定义。以该 Sign 工程为例, 源码中定义了多个宏, 您需要根据需求, 在编译时进行宏选择。单击 进行宏添加 (需在源码中定义)。
- 在 **Includes** 中进行头文件引用。CompassStudio 默认将工程的“inc”目录作为自定义的头文件引用目录。您也可以自定义目录, 如图 5-29 所示。

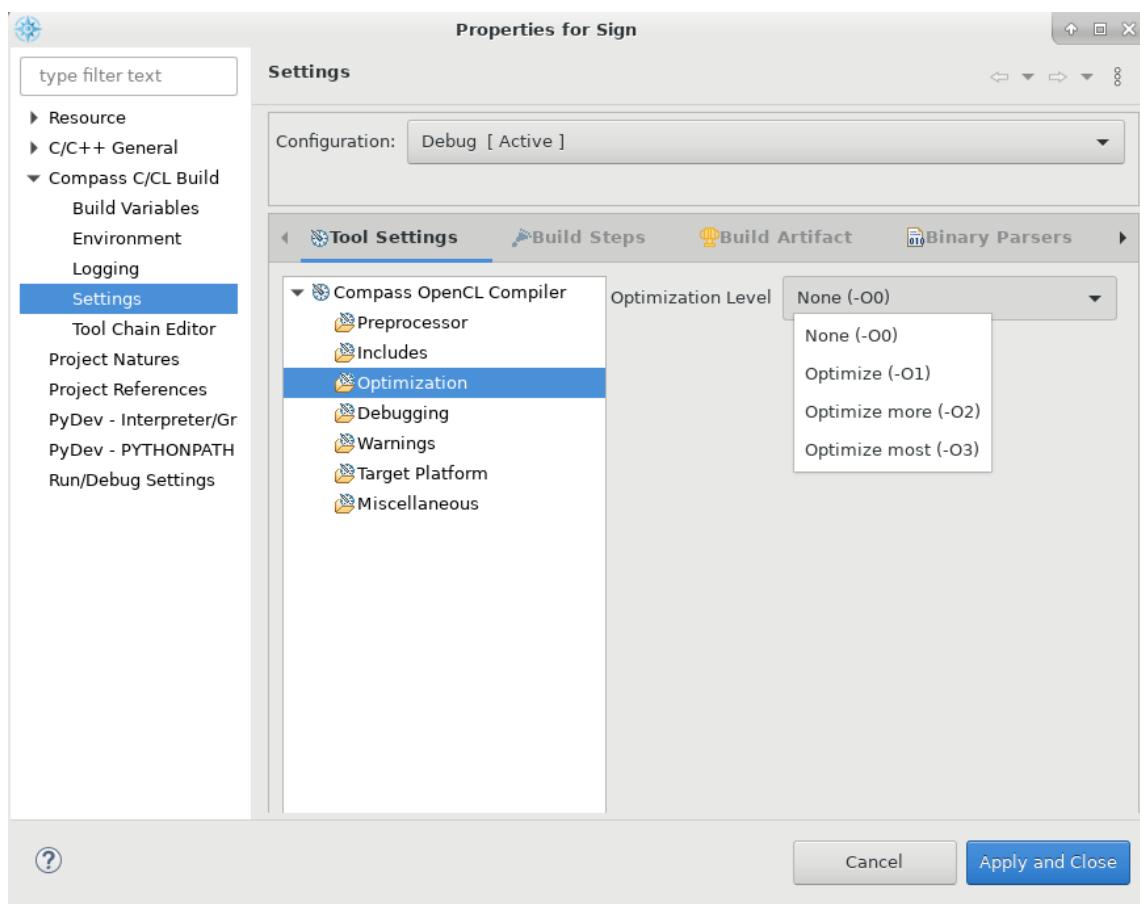
图 5-29: 配置 Compiler 自定义的 include 目录



在图 5-29 中, 红框内的 PATH 为配置 Compass OpenCL 的系统环境, 请勿删除。

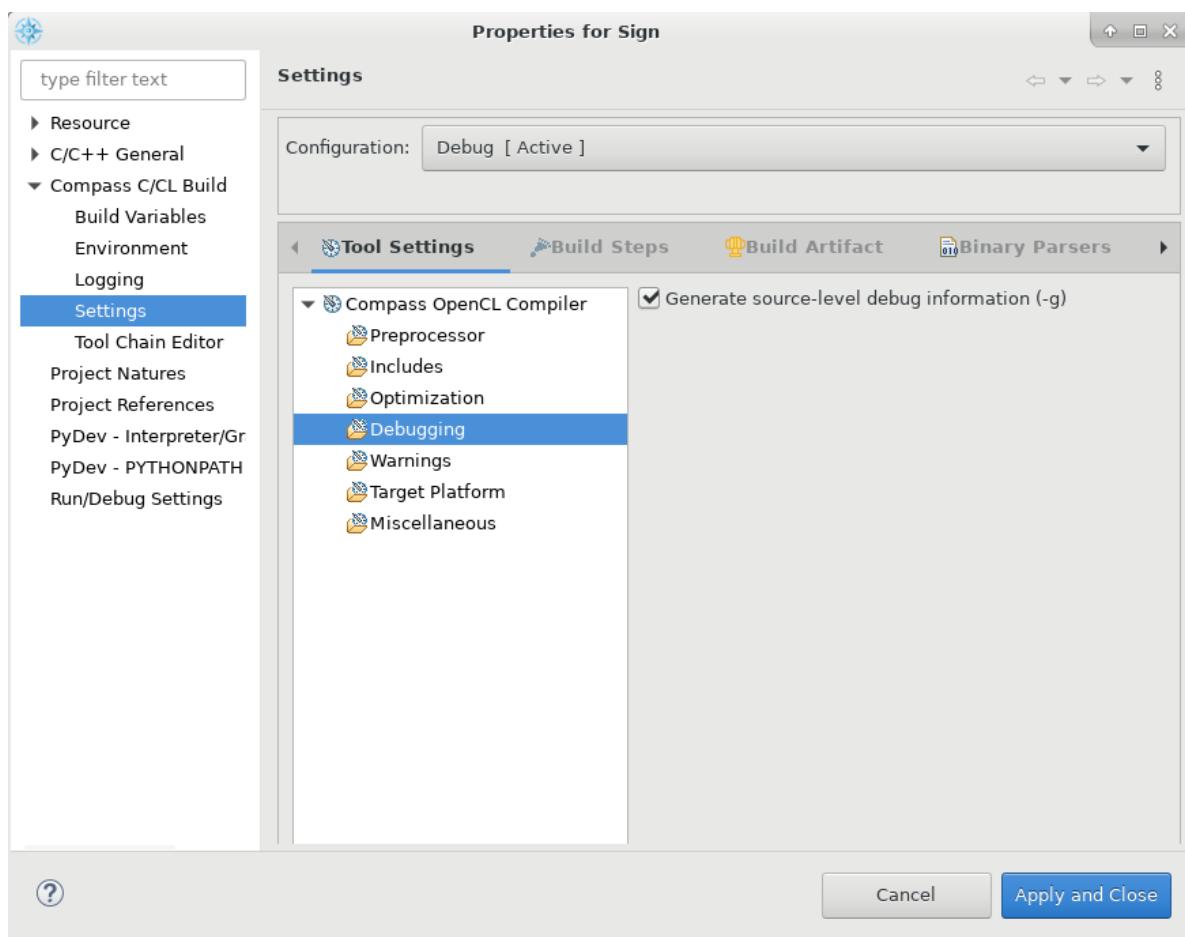
4. 在 Optimization 中可选择 Optimization Level, 如图 5-30 所示。

图 5-30: 配置 Compiler 的 Optimization Level



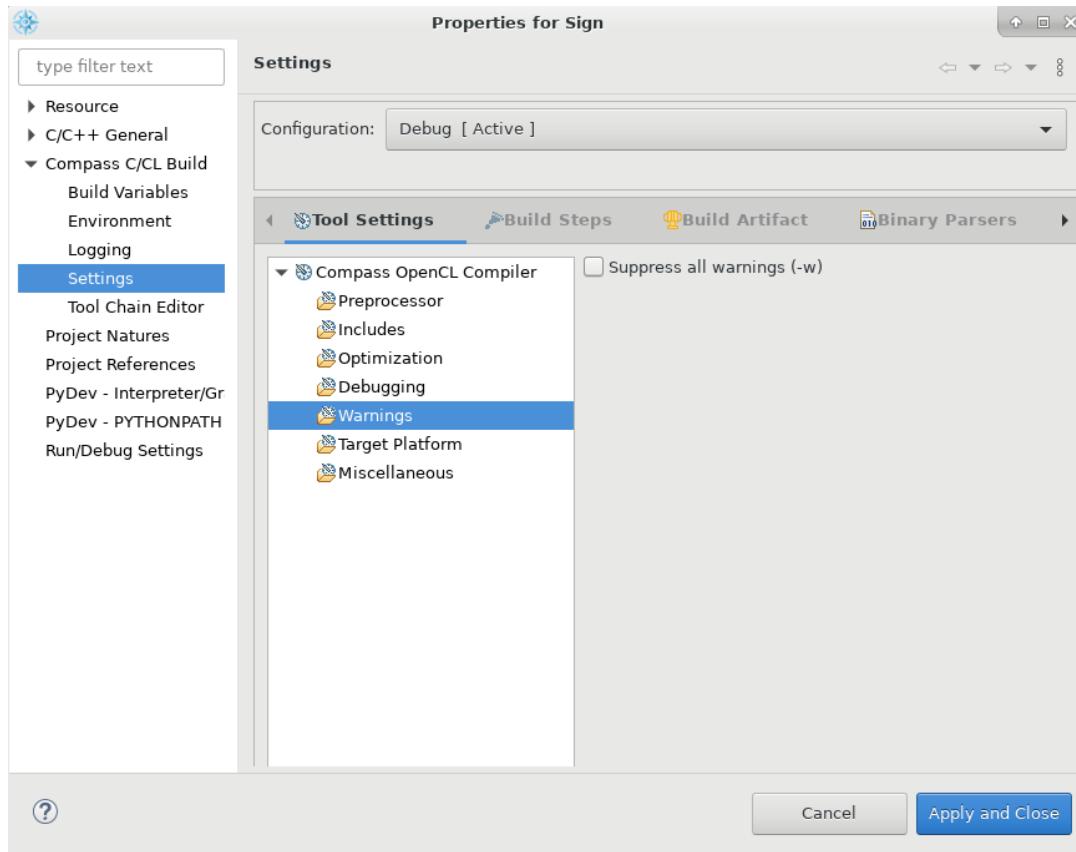
5. 在 Debugging 中选择调试参数, 如图 5-31 所示。

图 5-31: 配置 Compiler 的 Debugging 等级

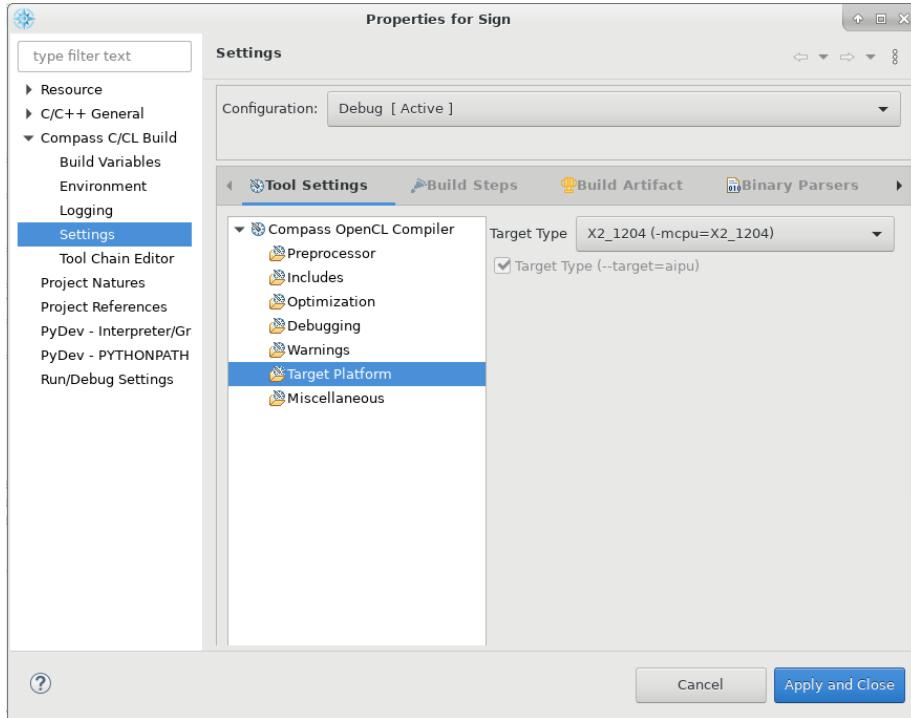


6. 在 Warnings 中可选择 warning 参数，如图 5-32 所示。

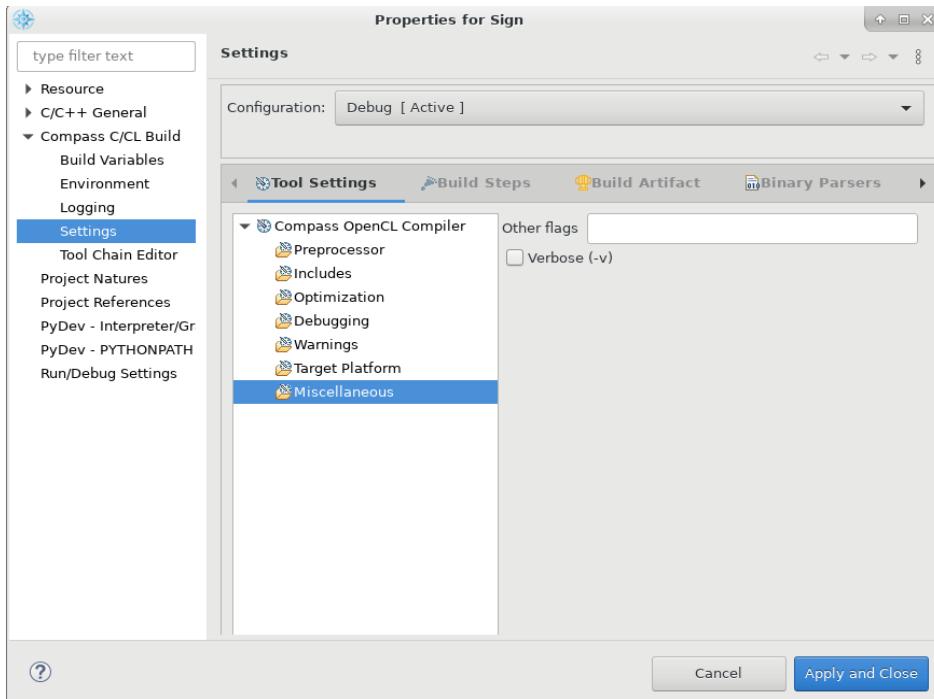
图 5-32: 配置 Compiler 的 Warning 参数



7. 在 Target Platform 中可选择 Target Type 等参数，如图 5-33 所示。

图 5-33: 配置 Compiler 的 Target

8. 在 Miscellaneous 中可选择其他编译参数，如图 5-34 所示。

图 5-34: 配置 Compiler 的其他编译参数

6 算子工程的编译及部署

本章节主要介绍 Compass C/OpenCL 算子工程的编译及部署。

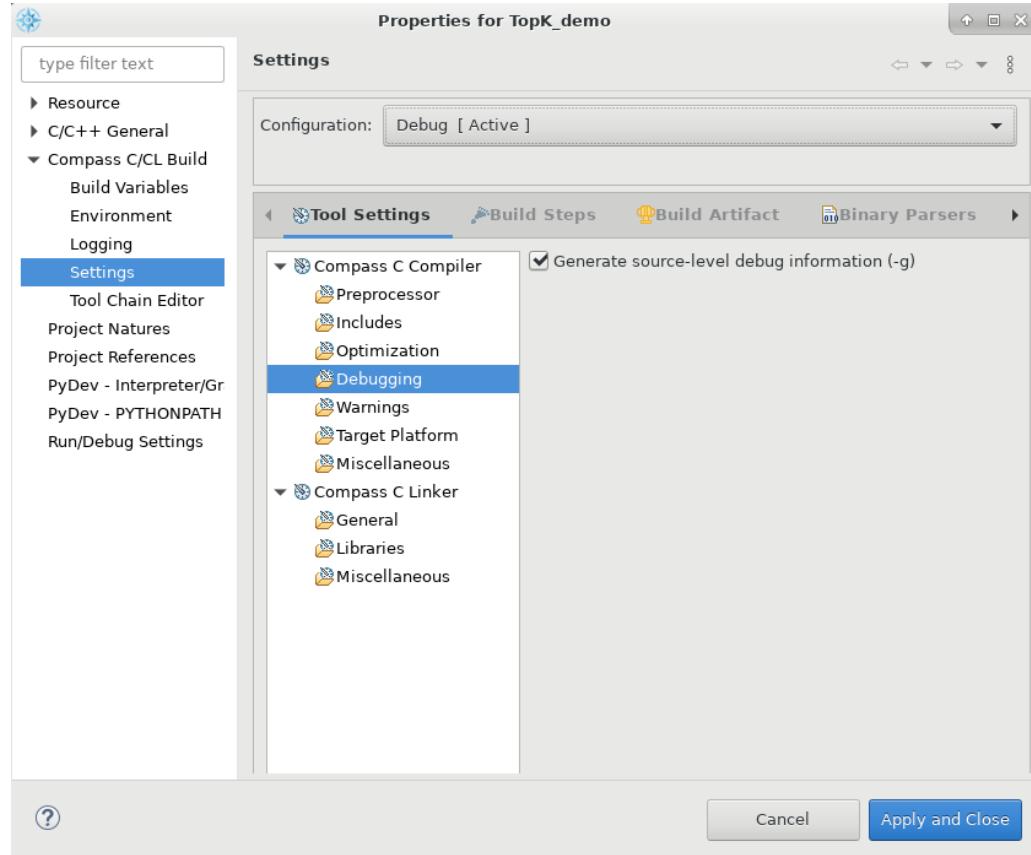
6.1 编译算子工程

根据 [5.1 创建算子工程](#) 和 [5.2 配置算子工程编译参数](#)配置完编译参数后，即可进行算子工程编译。

CompsStudio 中默认提供 Debug/Release Configuration 的编译参数管理：

- Debug Configuration 是方便您调试算子工程配置的编译参数（默认加“-g”），您也可以根据实际需求自行添加。
- Release Configuration 是方便您调试完算子后，正式部署的编译参数（默认不加“-g”），您也可以根据实际需求自行添加。

如图 6-1 所示，您可以选择 Debug 或者 Release Configuration 进行配置。

图 6-1: 编译参数 configuration 管理

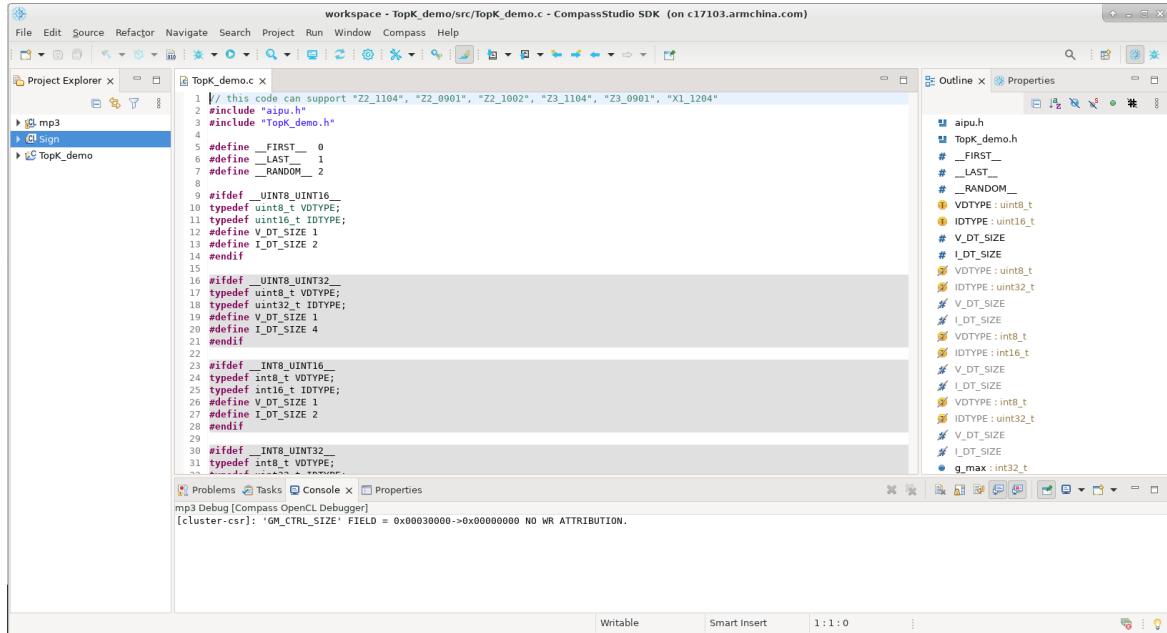
你可以根据实际情况，分别配置 Debug 和 Release Configuration 下的编译参数，请参考[5.2 配置算子工程编译参数](#)中的步骤进行配置。关于编译参数，请参考《Arm China Zhouyi Compass C/OpenCL Programming Guide》。

6.1.1 编译 Debug 版本的算子

Compass C 算子工程

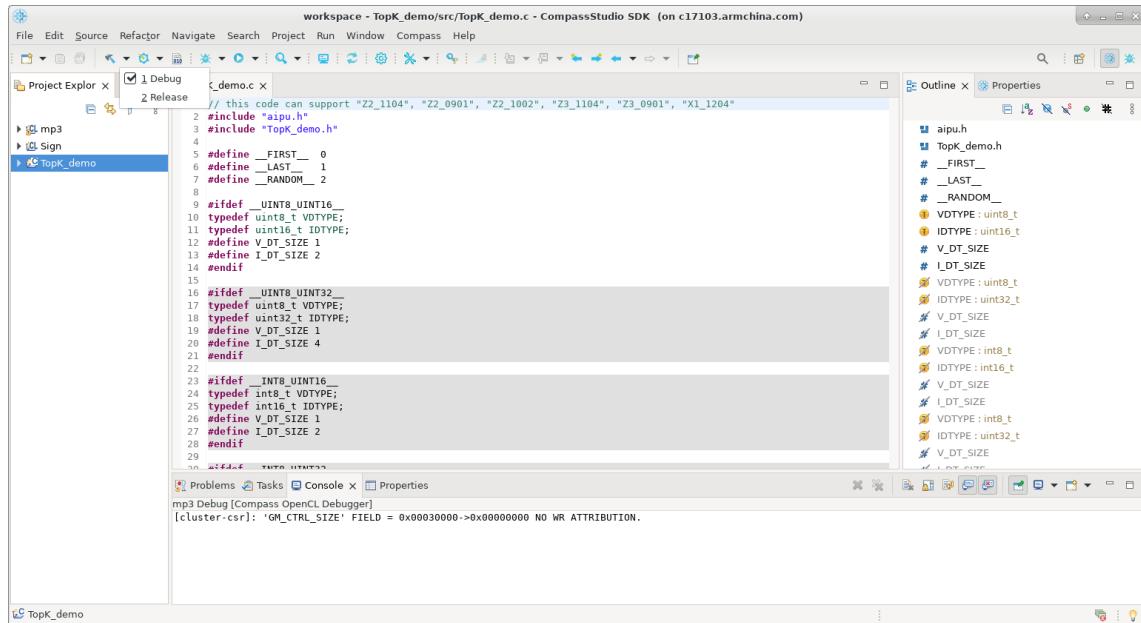
1. 配置完编译参数后，在 CompassStudio 主界面，选择需要编译的 configuration 配置，如图 6-2 所示。

图 6-2: 选择编译配置



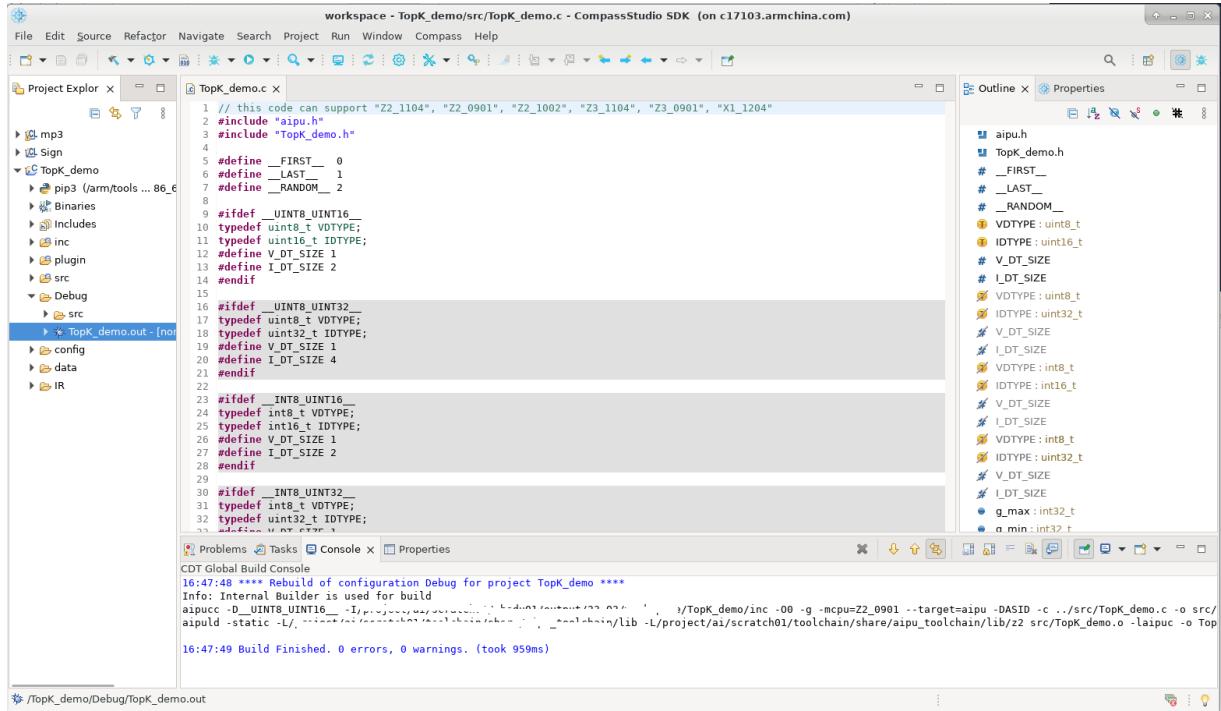
- 在图 6-2 中的界面上，选择 Debug 页签后，单击 图标进行编译，或者在 Build 下拉列表中，选择对应的 configuration 进行编译，如图 6-3 所示。

图 6-3: 编译 Debug 版的算子



- 单击 Build 图标进行编译，编译过程如图 6-4 所示。

图 6-4: 编译过程



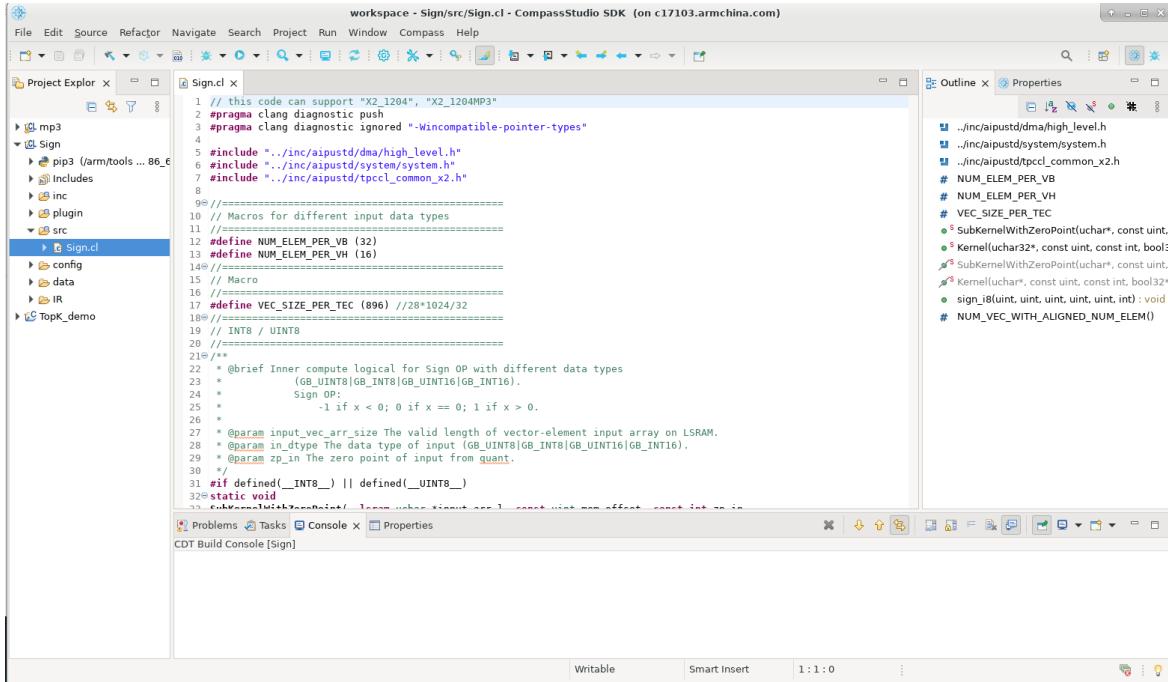
编译完成后，会在项目根目录下生成“Debug”目录，编译生成的 xxx.out elf 文件也位于该目录下。

4. 编译完成后，即可进行算子调试，请参考 [7 算子工程的调试](#)。

Compass OpenCL 算子工程

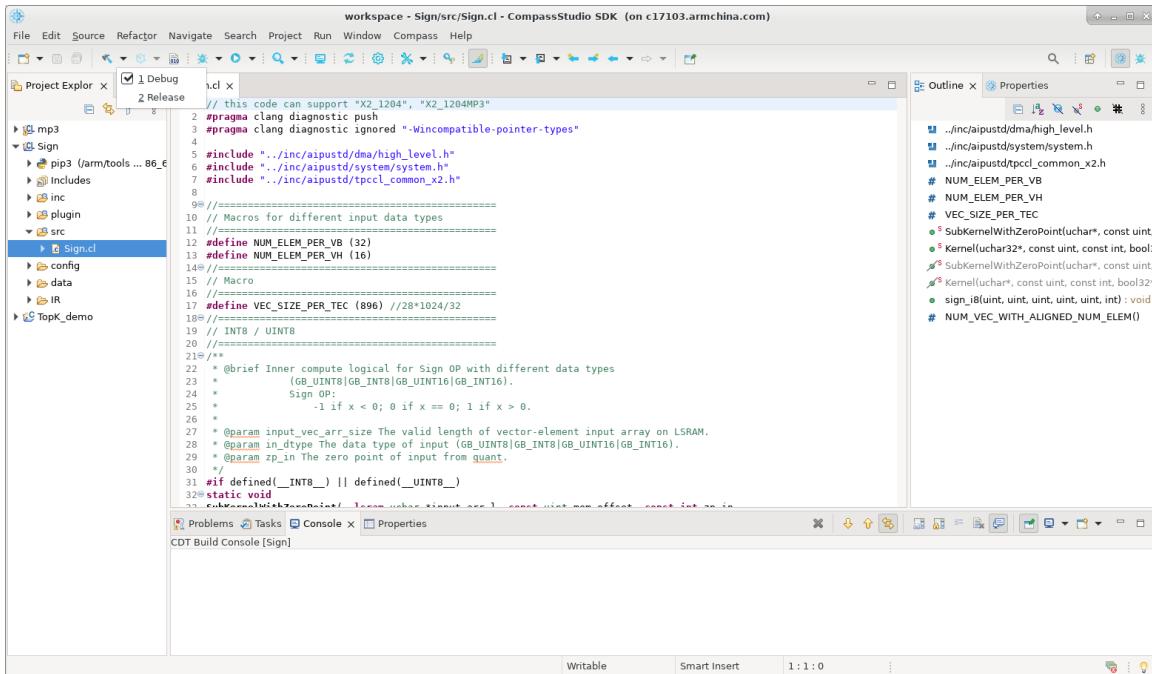
- 配置完编译参数后，在 CompassStudio 主界面，选择需要编译的 configuration 配置，如图 6-5 所示。

图 6-5: 选择编译配置



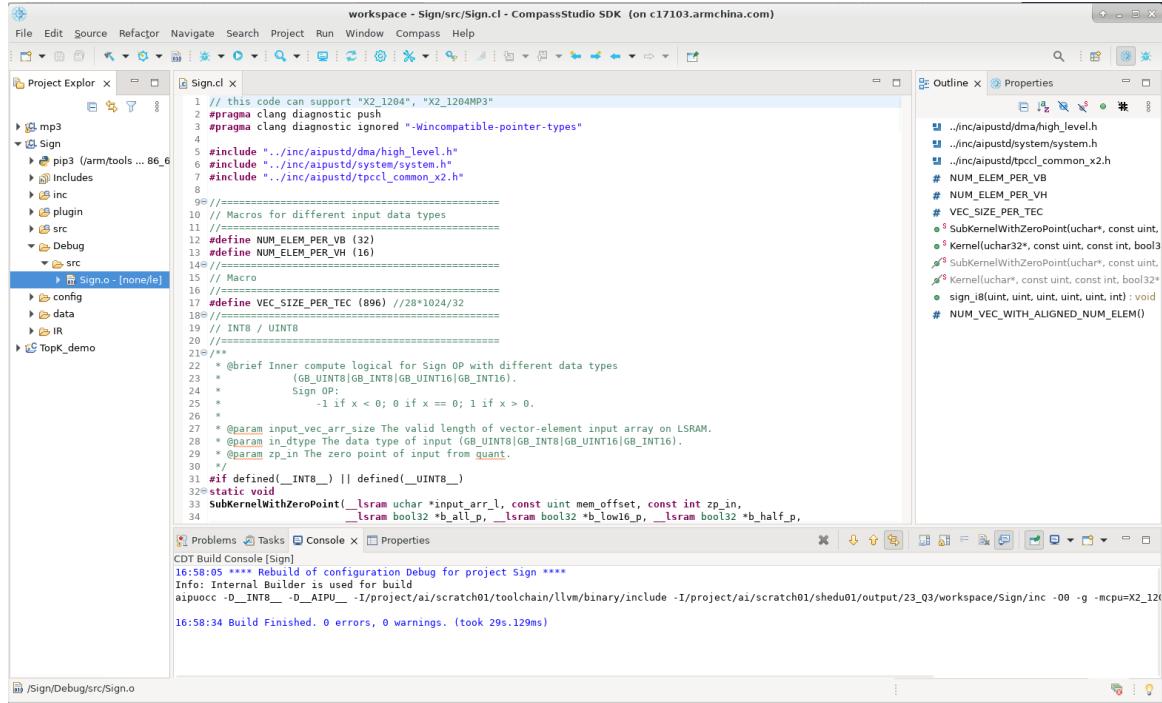
2. 在图 6-5 中的界面上，选择 Debug 页签后，单击 图标进行编译，或者在 Build 下拉列表中，选择对应的 configuration 进行编译，如图 6-6 所示。

图 6-6: 编译 Debug 版的算子



3. 单击 Build 图标进行编译，编译过程如图 6-7 所示。

图 6-7：编译过程



编译完成后，会在项目根目录下生成“Debug”目录，编译生成的 xxx.o 文件也位于“Debug/src”目录下。

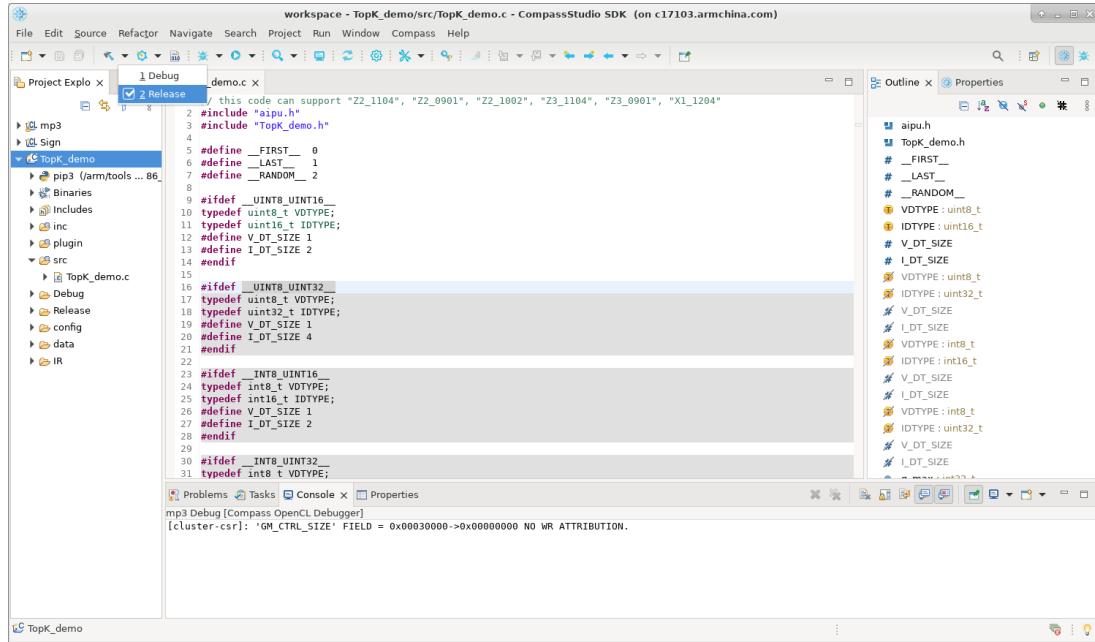
4. 编译完成后，即可进行算子调试，请参考 [7 算子工程的调试](#)。

6.1.2 编译 Release 版本的算子

Compass C 算子工程

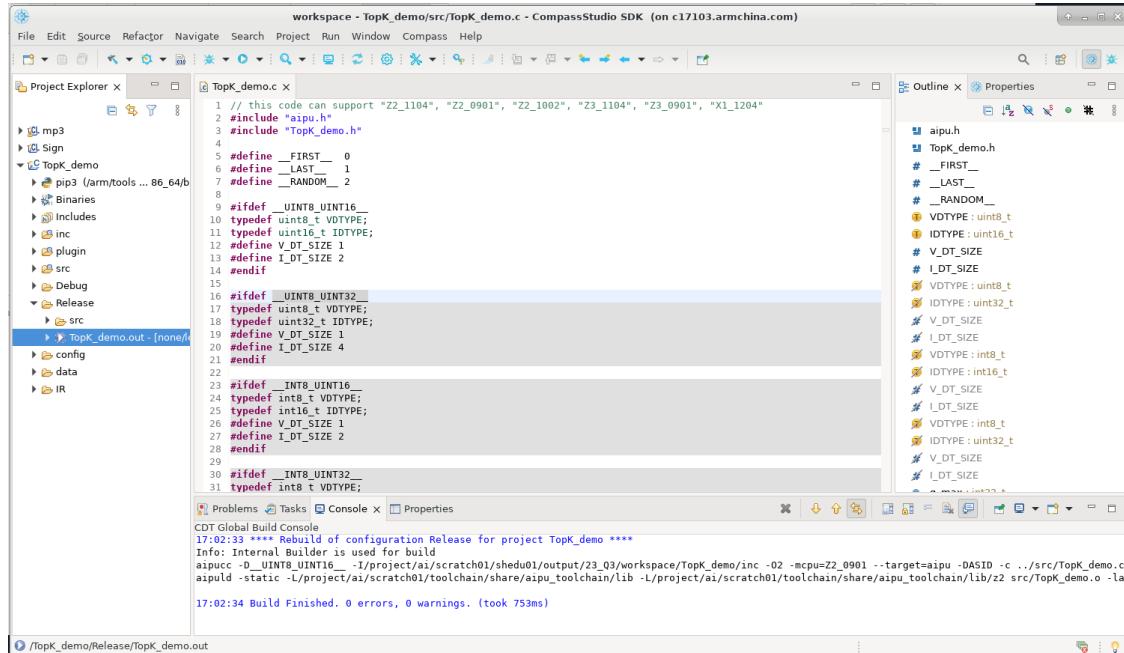
1. 算子工程调试完成后，可进行算子 Release 版本的编译。单击 Build 图标，选择 Release 进行编译，如图 6-8 所示。

图 6-8: 编译 Release 版的算子



2. 查看 Release Configuration 编译过程, 如图 6-9 所示。

图 6-9: 编译过程



编译完成后,会在项目根目录下生成 Release 目录,编译后的 xxx.out elf 文件也位于该目录下。

至此，Compass C 算子工程 Debug/Release 版本的编译已经完成。

Compass OpenCL 算子工程

1. 算子工程调试完成后，可进行算子 Release 版本的编译。单击 Build 图标，选择 Release 进行编译，如图 6-10 所示。

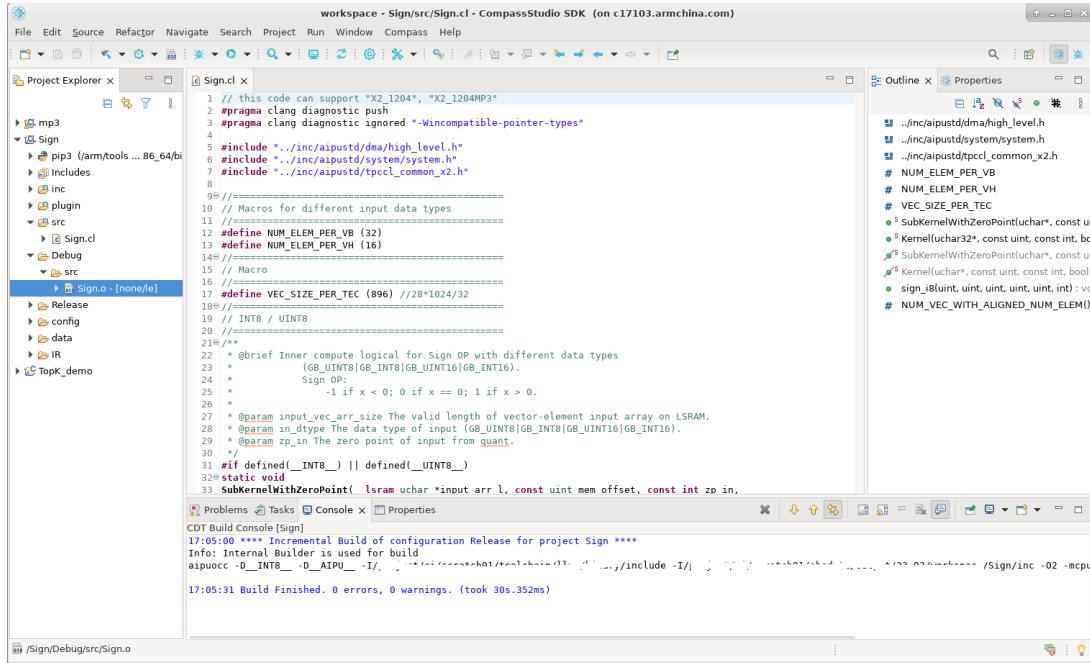
图 6-10：编译 Release 版的算子

The screenshot shows the CompassStudio SDK interface with the following details:

- File Menu:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Compass, Help.
- Project Explorer:** Shows the project structure with branches for mp3, Sign, and TopK_demo. The Sign branch is expanded, showing sub-folders like Includes, inc, plugin, src, and sub-folders for Sign, Debug, config, data, IR, and Release.
- Sign.c File Content:** The code defines various macros and includes for different data types. It includes headers for clang diagnostic, aiupstd/dma/high_level.h, aiupstd/system/system.h, and aiupstd/tpcl_common_x2.h. Macros define NUM_ELEM_PER_VB (32), NUM_ELEM_PER_VH (16), VEC_SIZE_PER_TEC (896), and INT8 / UINT8. The code also handles sign operations and vector-element input arrays.
- Outline View:** Shows a list of symbols and includes from the Sign.c file.
- Properties View:** Shows build configurations for the project.
- Bottom Status Bar:** Displays "Build Project: (41%)".

2. 查看 Release Configuration 编译过程，如图 6-11 所示。

图 6-11: 编译过程



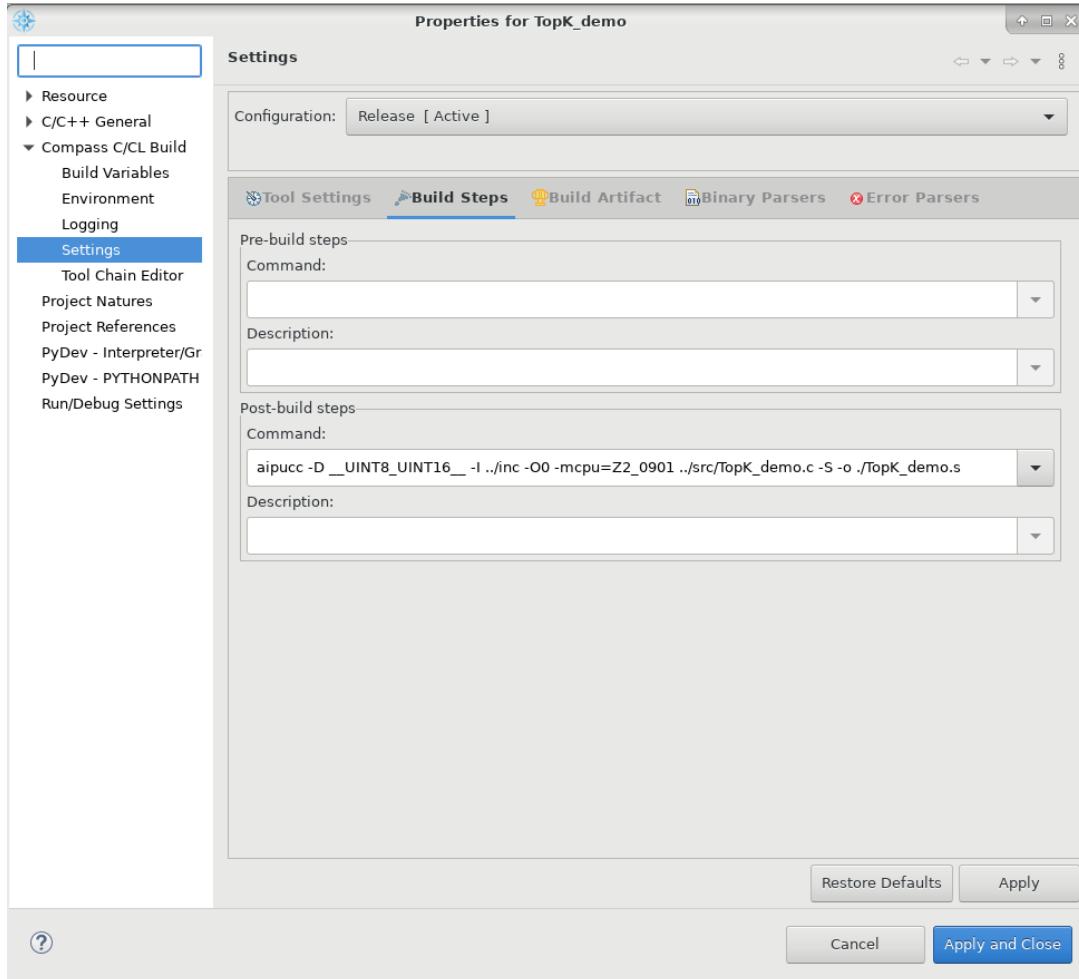
编译完成后，会在项目根目录下生成 Release 目录，编译后的 xxx.o 文件也位于“Release/src”目录下。

至此，Compass OpenCL 算子工程 Debug/Release 版本的编译已经完成。

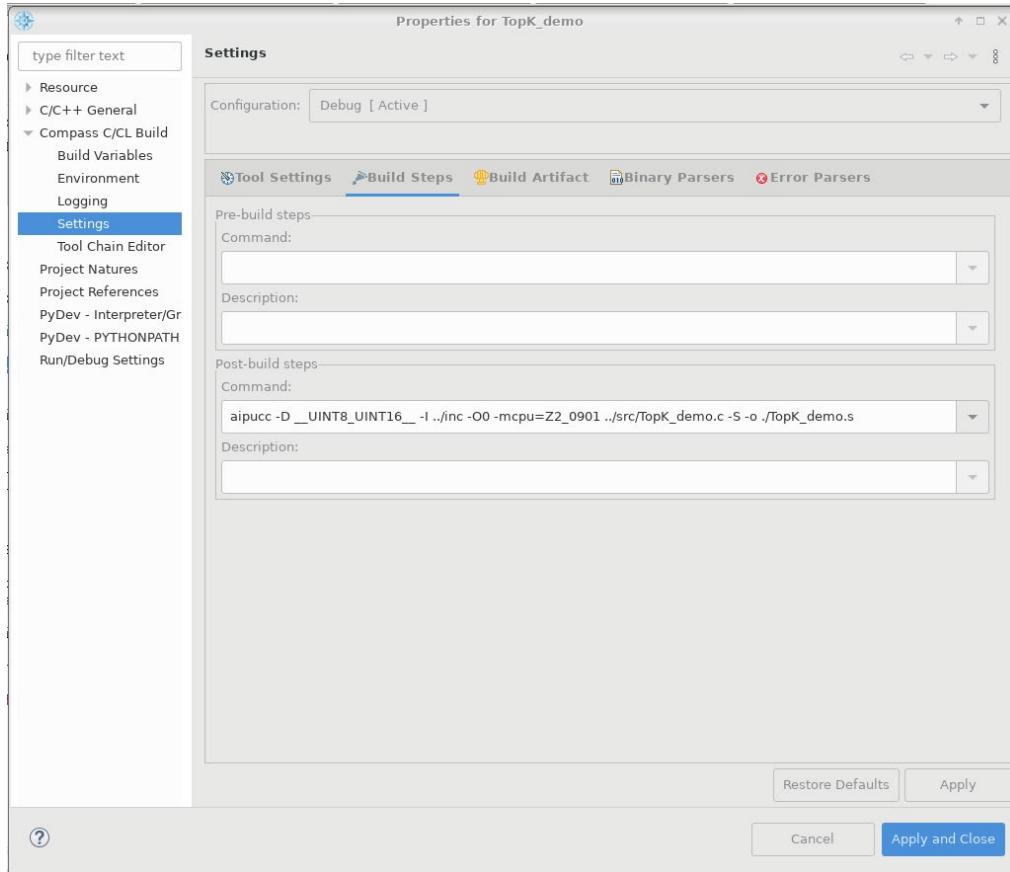
6.1.3 编译生成汇编文件

Compass C 和 Compass OpenCL 算子工程生成汇编文件的步骤一致，具体步骤如下：

- 参考配置 Assembler 参数的操作，以 Debug Configuration 为例，配置如图 6-12 所示。

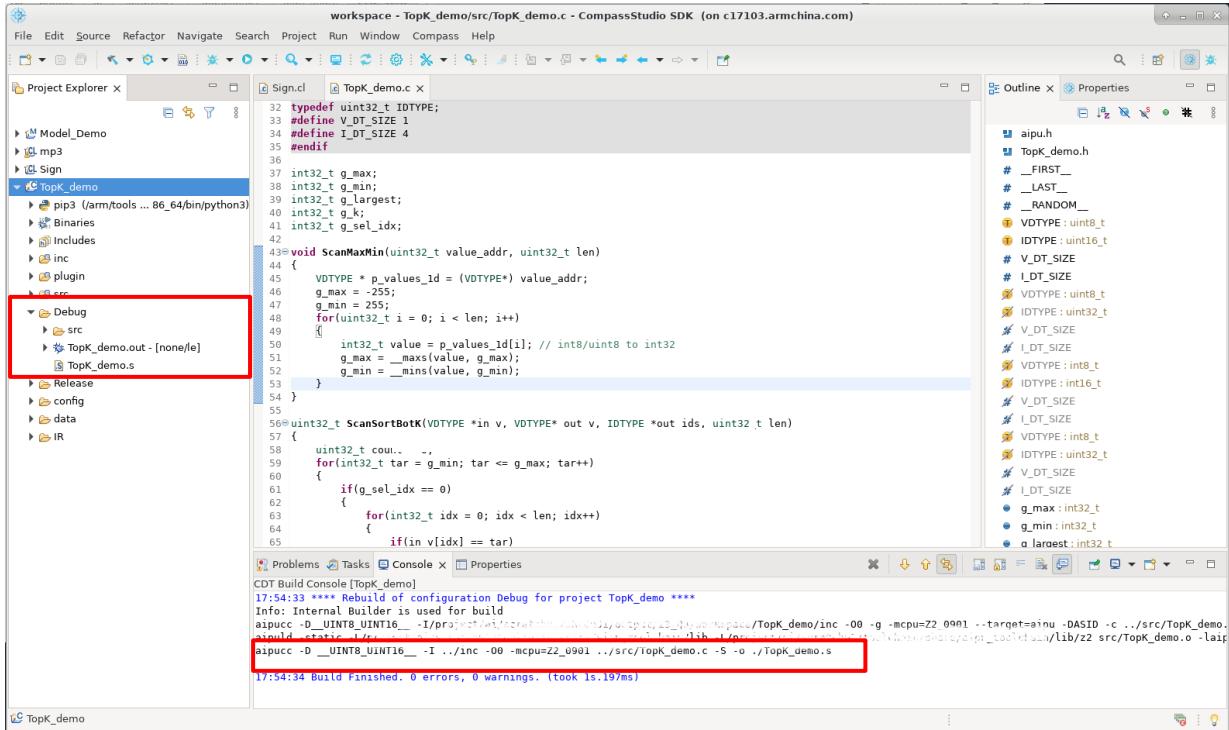
图 6-12: 汇编参数配置

2. 查看当前执行路径，在上图界面中选择 **Environment > Configuration > CWD**，如图 6-13 所示。

图 6-13: 查看执行环境

- 回到 CompassStudio 主界面，单击 Build 图标，选择对应的 configuration 进行编译，如图 6-14 所示。

图 6-14: 编译产生汇编文件



4. 回到 CWD 对应的目录下查看汇编文件即可。

6.2 编写算子 plugin

CompassStudio 默认用户将 plugin 脚本放置在工程根目录下的 Plugin 文件夹。Plugin 的编写，请参考 CompassStudio 提供的 sample 模块进行编写。

编写 plugin 前，请仔细阅读《Arm China Zhouyi Compass NN Compiler User Guide》的“Supporting customized operators”章节。

Compass C 算子工程

Compass C 算子提供的 sample 中包含 Parser, Optimizer, GBuilder 和 Checker 的 plugin。其中 GBuilder plugin (aipu_lib_xxx.py)，需要修改插件内容后才可使用。请确保“generate_code_name”和编译产生的.out 文件名一致。参考如下：

```
def generate_code_name(self, sgnode, nodes):
    name = "xxx.out"
    return name
```

Compass OpenCL 算子工程

Comps OpenCL 提供的 sample 中包含 GBuilder 和 Checker 的 plugin。其中 GBuilder plugin (aipu_lib_xxx.py)，需要修改插件内容后才可使用。请确保“generate_code_name”和编译产生的 .o 文件名一致。参考如下：

```
def generate_code_name(self, sgnode, nodes):
    name = "xxx.o"
    return name
```



算子调试前，需要将算子 plugin 编写完成。因为在算子调试时，会运行 Compass IR（算子调试时，CompassStudio 会自动对算子和 plugin 进行临时部署），若 plugin 没有编写完成，Compass IR 无法加载该算子。

6.3 部署算子工程

算子工程调试完成、编译 Release 版本完成、以及算子 plugin 编写完成后，需要对算子进行正式部署。Compass C 和 Compass OpenCL 算子部署、校验的步骤一致，本小节以 Compass C 算子部署、校验为例，供 Compass OpenCL 算子部署、校验参考（两种部署方式的唯一区别为：Compass C 部署 .out , Compass OpenCL 部署 .o ）

CompassStudio 对算子的部署分为两类：

- 部署到指定的 NN-Model 工程，即该算子只对当前模型工程有效。
- 部署到系统环境变量（参考 [2.3 CompassStudio 开发环境](#)配置算子部署的环境），即该算子对所有模型工程有效。

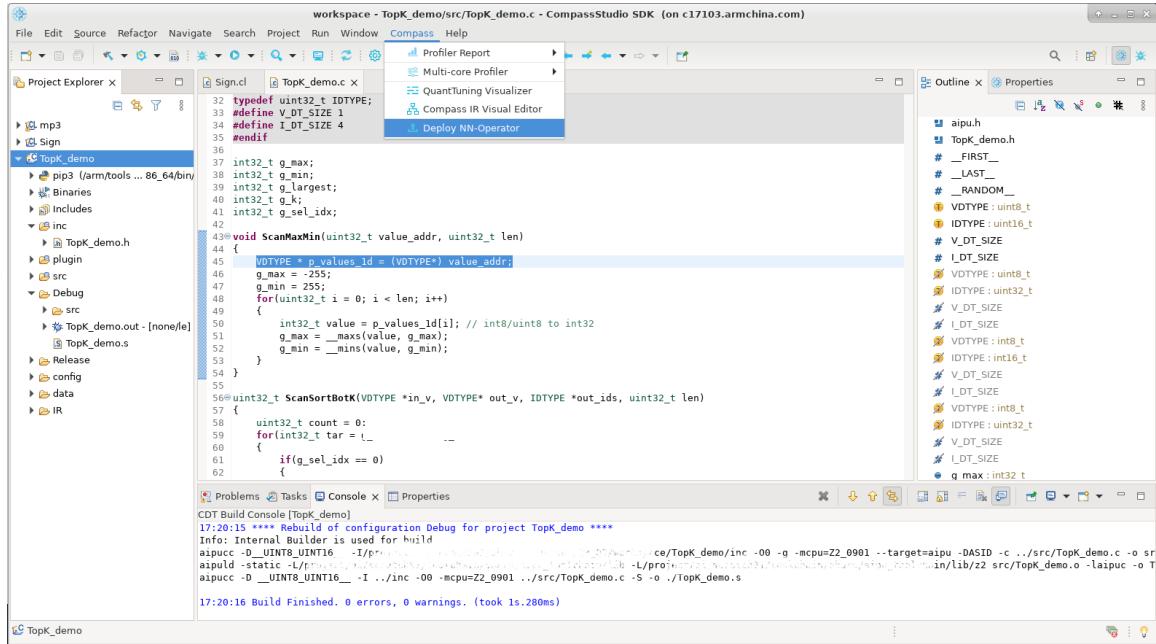


部署算子工程前，请确保 NN-SDK 和 Python 环境配置正确。

6.3.1 部署到指定的模型工程

1. 选择需要部署的算子工程（以 TopK 工程为例），选择 Compass > Deploy NN-Operator，如图 6-15 所示。

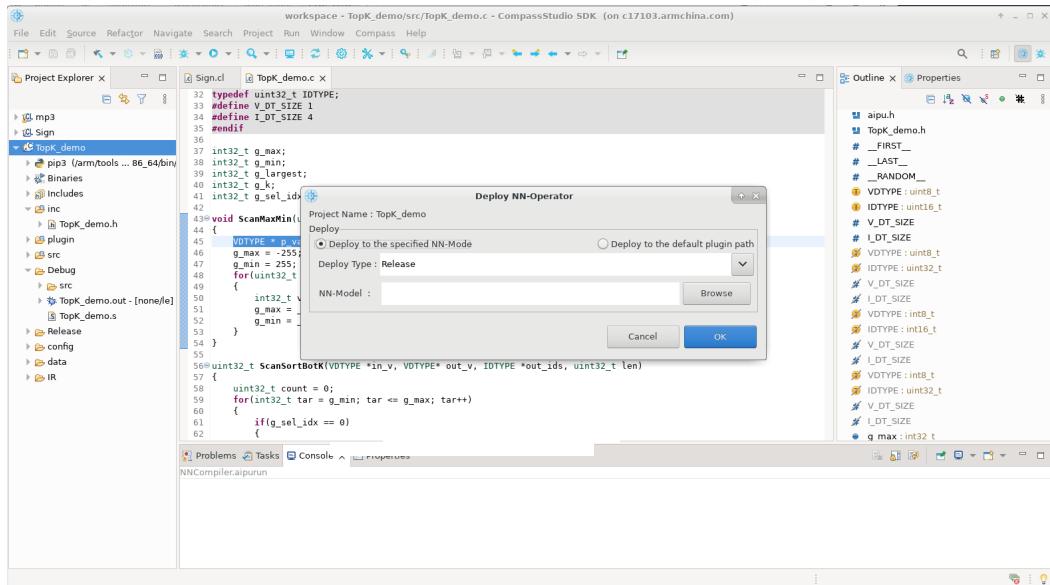
图 6-15: 算子部署操作



您也可以右键单击工程，然后选择 Deploy NN-Operator。

- 在算子部署对话框中，选择 Deploy to the specified project，如图 6-16 所示。

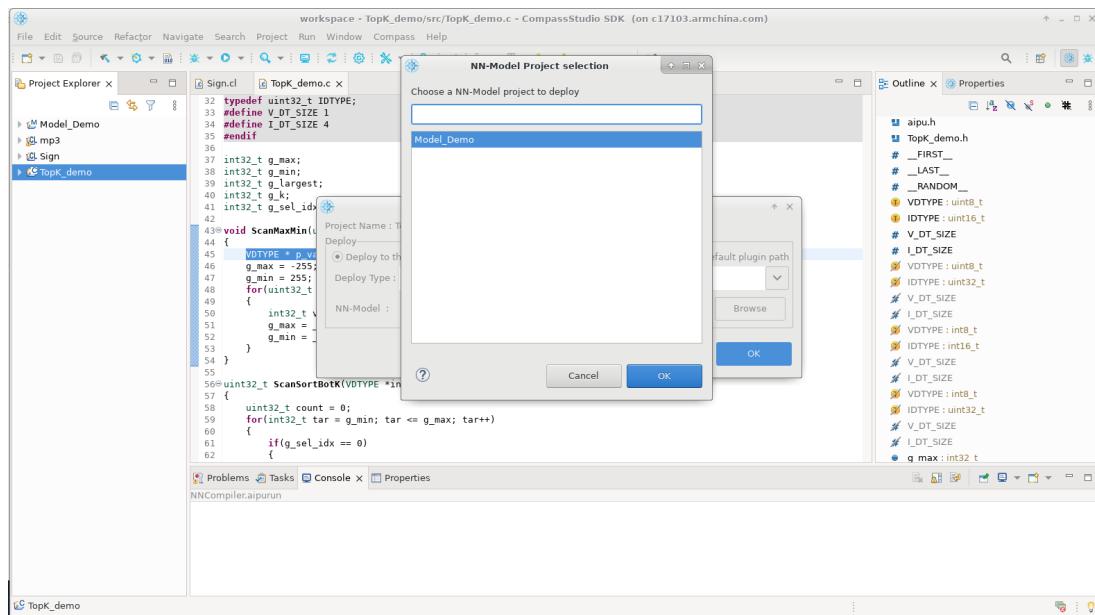
图 6-16: 部署到指定模型工程



- 在 Deploy Type 下拉列表，选择部署算子的类型。CompassStudio 默认选择 Release，即部署 Release 版本的算子。

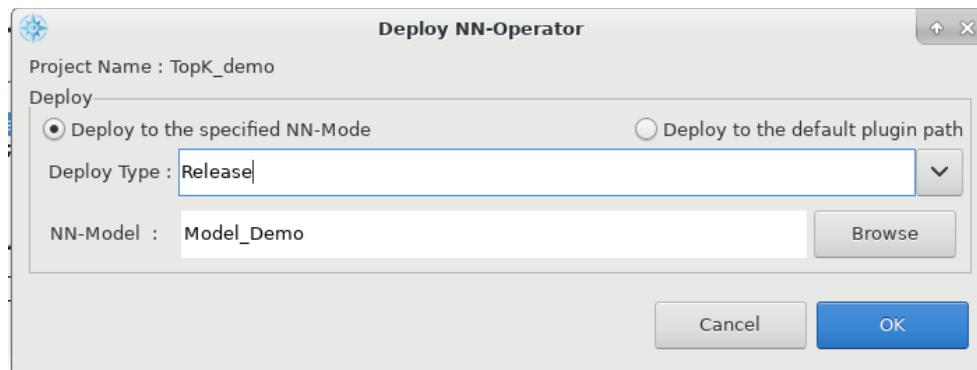
4. 单击 **Browse**, 可加载当前 workspace 中的 NN-Model 进行部署, 如图 6-17 所示。

图 6-17: 选择部署的模型工程



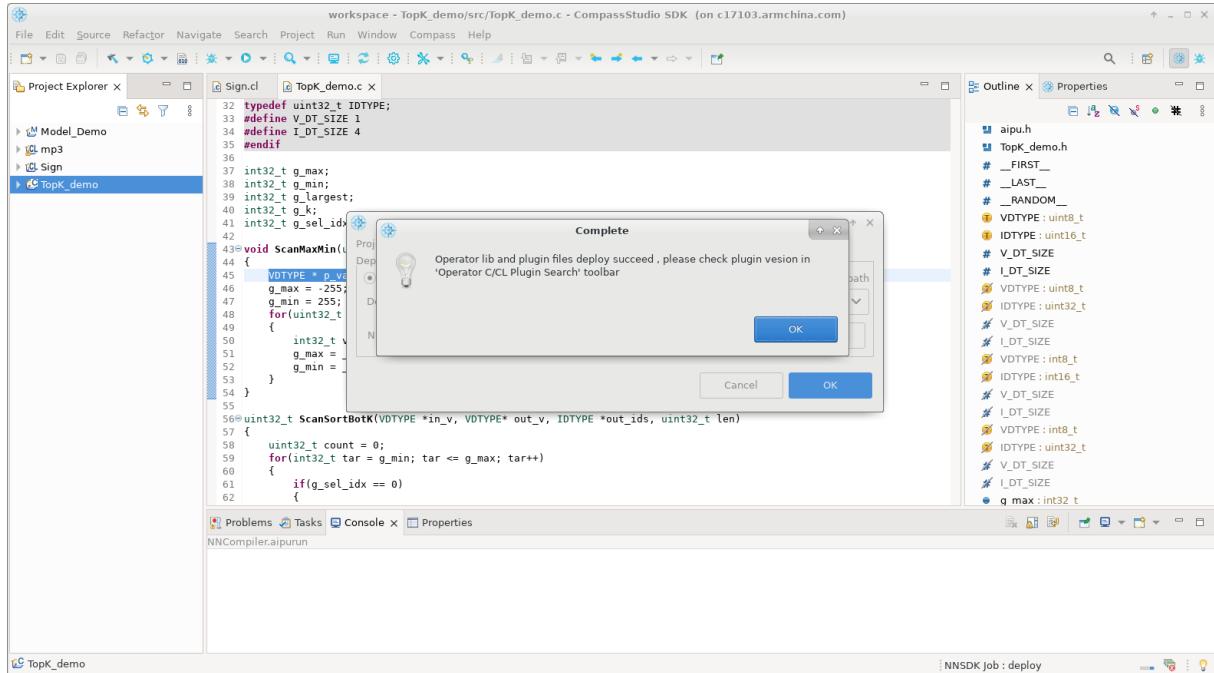
5. 选择想要部署的 NN-Model project (以 Model_demo 为例), 单击 **OK**, 如图 6-18 所示。

图 6-18: 选择部署的模型工程



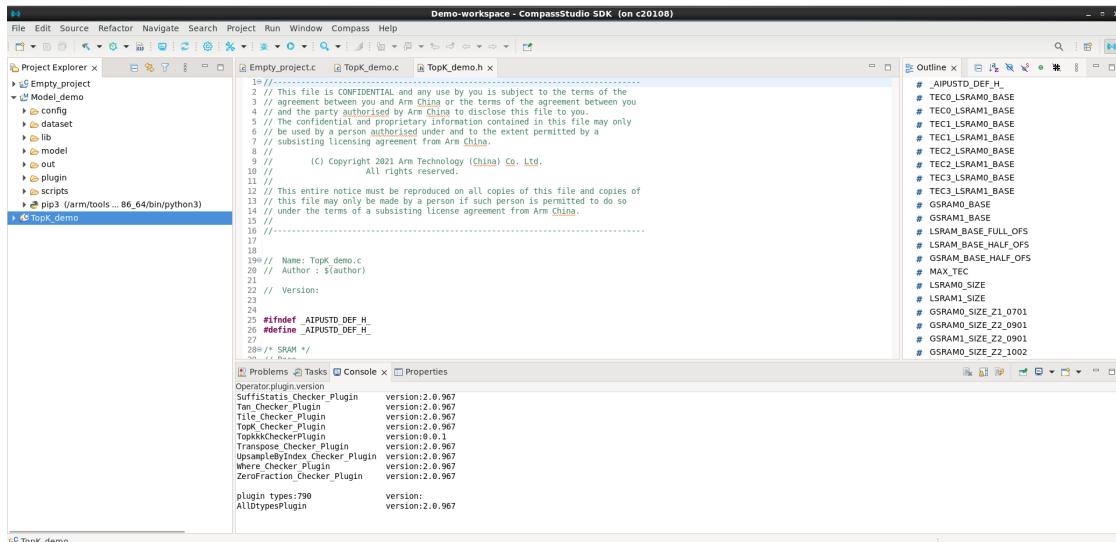
6. 在图 6-18 所示的对话框中, 单击 **OK** 开始部署, 过程如图 6-19 所示。

图 6-19: 部署过程



- 部署成功后，可以打开“Operator C plugin search”对话框，对算子部署成功与否进行检查。请参考 6.3.3 算子 Plugin 校验。
- 部署成功后，打开部署的模型工程，会发现多了 lib（算子部署路径）和 plugin（plugin 部署路径）目录。如图 6-20 所示。

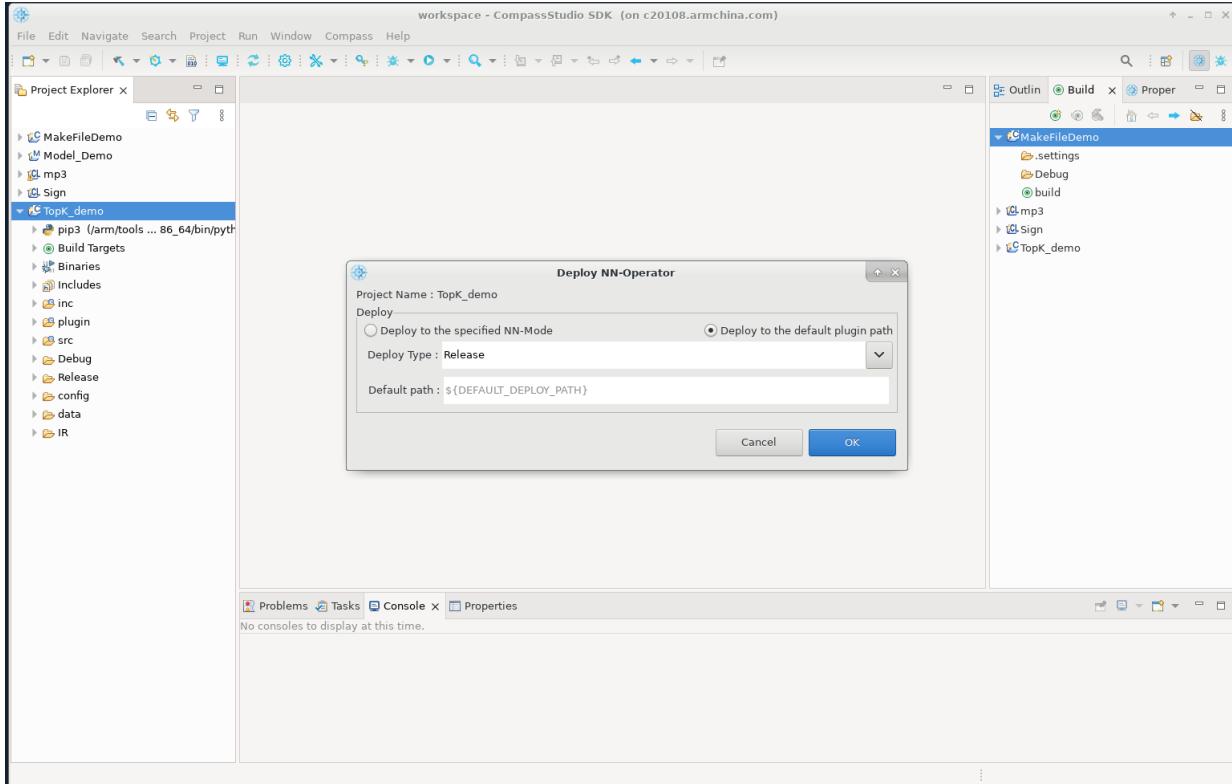
图 6-20: 成功部署到指定模型工程



6.3.2 部署到系统环境

- 在图 6-16 所示的对话框中, 选择 Deploy to the default plugin path, 如图 6-21 所示。

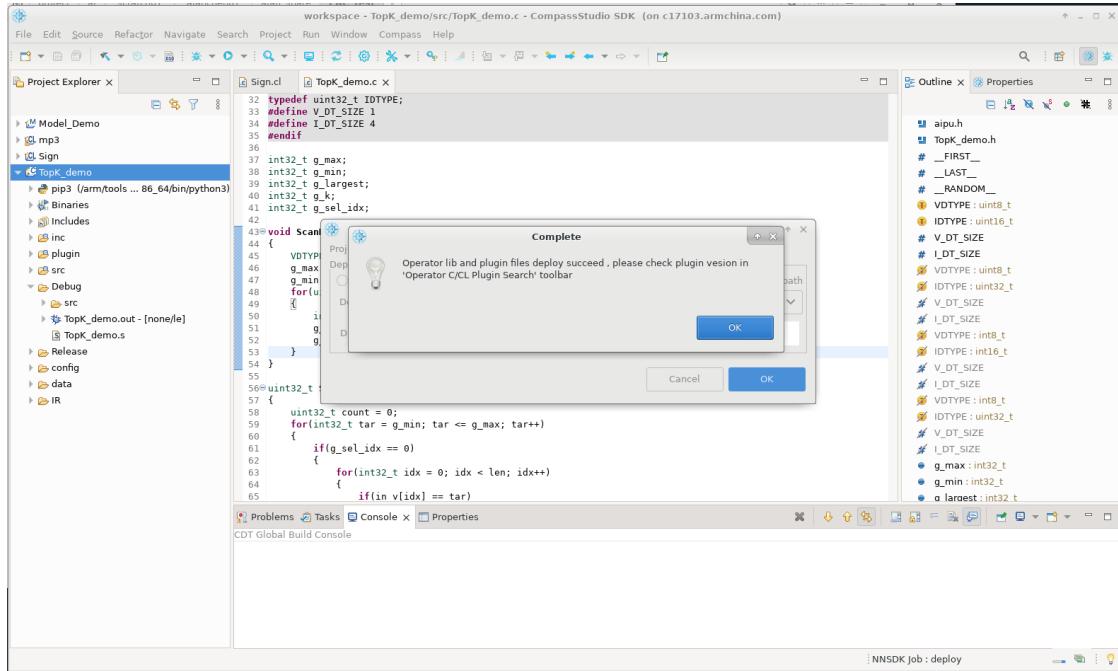
图 6-21: 部署到系统环境



请确认已按照 [2.3 CompassStudio 开发环境](#) 正确配置算子部署环境。默认部署到该地址环境中。

- 在图 6-21 所示的对话框中, 单击 OK 开始部署, 过程如图 6-22 所示。

图 6-22: 部署过程

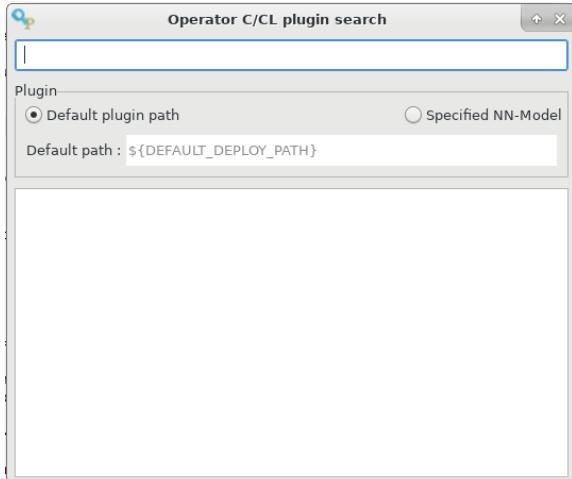


- 部署成功后，可以打开“Operator C plugin search”对话框，对算子部署成功与否进行检查。请参考 [6.3.3 算子 Plugin 校验](#)。
- 也可以在 [2.3 章节](#)中算子部署的路径下查看。

6.3.3 算子 Plugin 校验

算子部署成功后，需要单击 图标打开“Operator C/CL plugin search”对话框进行校验。Plugin 校验分为指定模型工程 Plugin 校验和默认系统环境校验，如图 6-23 所示。

图 6-23: Plugin 校验



- 在图 6-23 所示的对话框中, 选择 **Default plugin path**, 输入部署的 Plugin 名称, 然后按 **Enter** 键进行查询, 如图 6-24 所示。

图 6-24: 系统环境 Plugin 校验

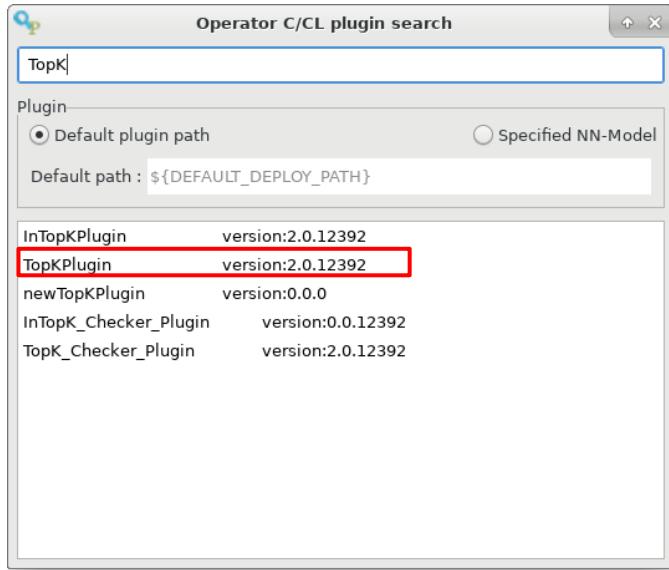


图 6-24 中的“TopKPlugin”即为部署到系统环境中的算子 plugin。

- 在图 6-23 所示的对话框中, 选择 **Default plugin path**, 输入部署的 Plugin 名称, 然后按 **Enter** 键进行查询, 如图 6-25 所示。

图 6-25: 指定模型工程 Plugin 校验

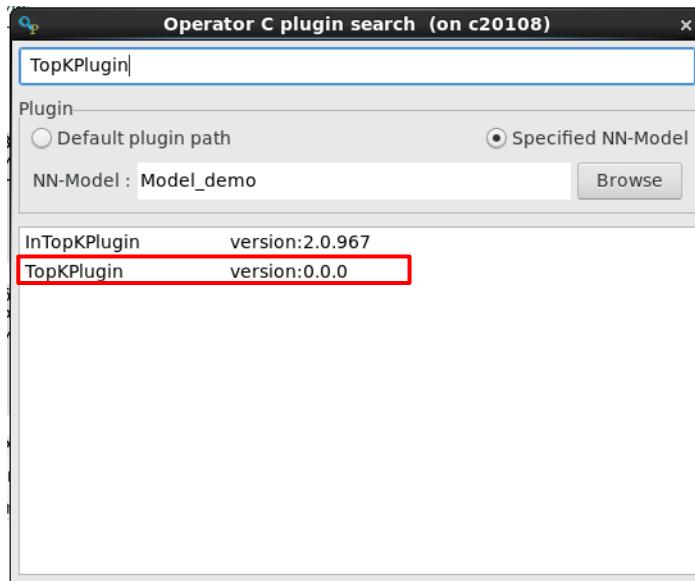


图 6-25 中的“TopKPlugin”即为部署到指定模型工程的算子 plugin。

7 算子工程的调试

Compass C 和 Compass OpenCL 算子工程的调试步骤一致。本小节以 Compass C 算子调试为例，供 Compass OpenCL 算子调试参考。

CompassStudio 支持的调试模式分为 Simulator 调试和 Hardware（开发板）调试。

- Simulator 调试不需要连接开发板，配置 Simulator 相关软件环境即可进行调试。
- Hardware 调试需要使用 Jtag 连接开发进行调试，同时，需要在开发板上开启 aipudbgserver。

Compass_zhouyi 开发板支持 Jtag 调试。数据传输支持局域网络传输，可以使用网线进行数据传输，也可以使用存储设备自行拷贝，即开发板支持 Debug on hardware with network 和 Debug on hardware with no network（请参考《Arm China Zhouyi Compass Debugger User Guide》）。

目前该版本的 CompassStudio：

- 支持断点调试的相关操作，如 Step into、Step over、Step return、Resume 等。
- 支持 Variables、Memory、Register、Disassembly、Expression、Vector Display、Watchpoint 以及 Value Change（不支持向量寄存器值的修改）等功能。
- 支持周易 X1 单核调试。
- 支持周易 X2 单核，多 TEC 异步调试。
- 支持周易 X2 多核多 TEC 异步调试。
- 不支持周易 X2 单核多 TEC 同步调试和多核多 TEC 同步调试。

7.1 Debug on Simulator

Compass C 和 Compass OpenCL Debug on Simulator 的步骤一致，调试页面也基本一致，本小节将分别介绍其使用方法。

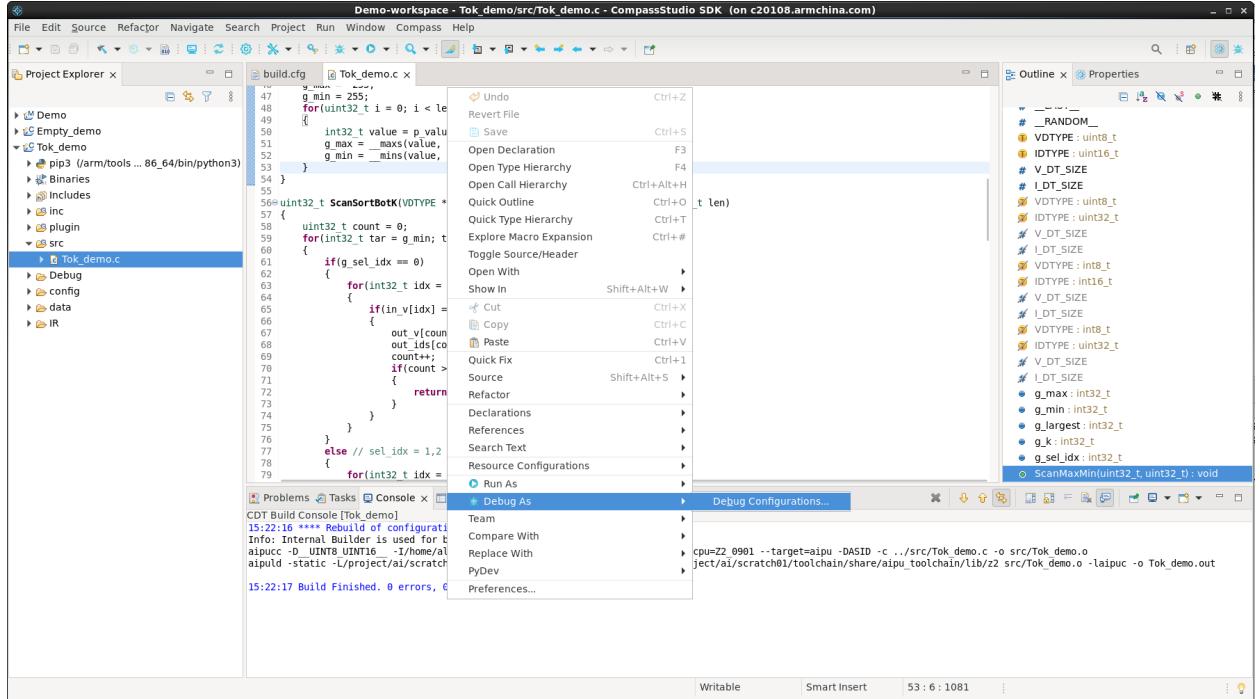
参考 [6.1 编译算子工程](#)，由 Compass C/OpenCL 工程，生成可以调试的“xxx.out/xxx.o”文件。参考 [6.2 编写算子 plugin](#)，完成 plugin 脚本编写。

Simulator 调试步骤请参考：

7.1.1 Compass C Debug on Simulator

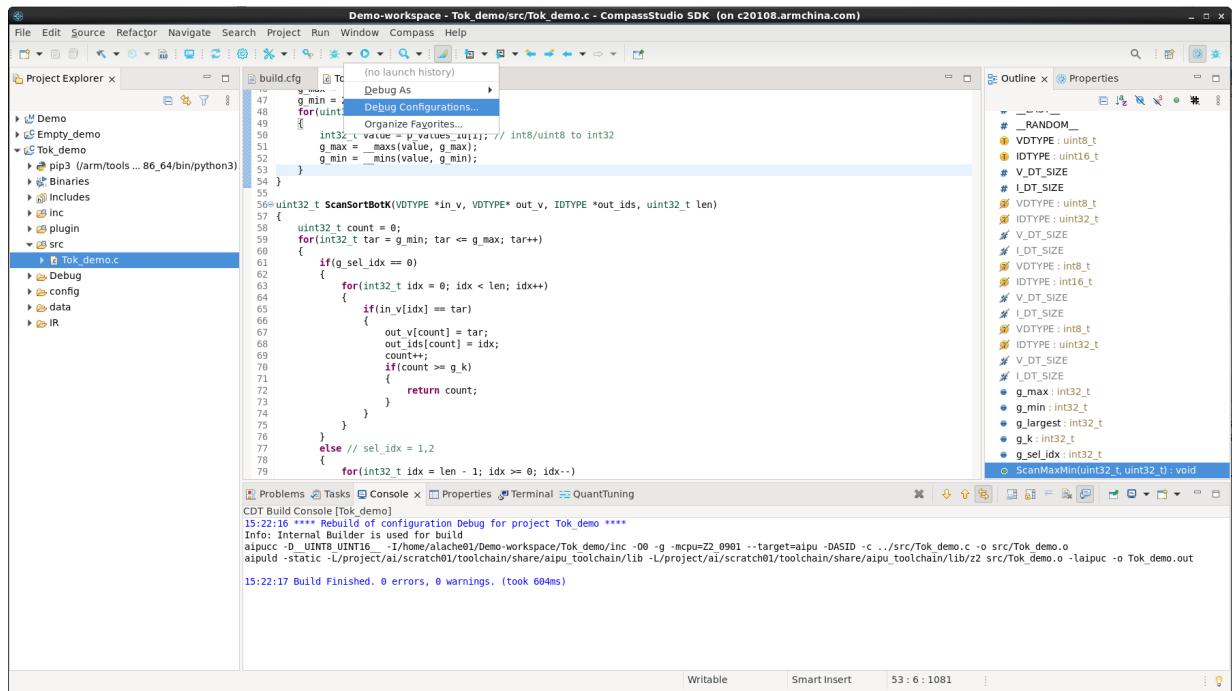
- 右键单击算子 .c 文件，然后选择 Debug As > Debug Configurations，如图 7-1 所示。

图 7-1：调试操作



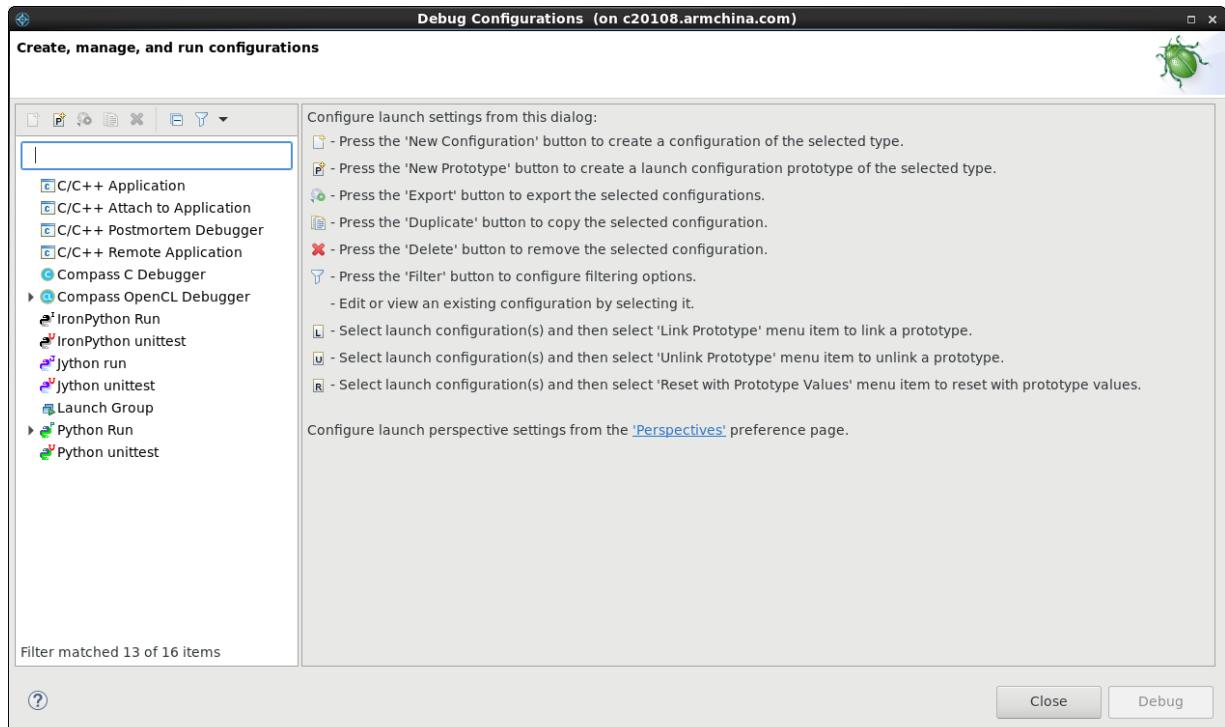
或者在 CompassStudio 工具栏，单击 Debug 图标，然后选择 Debug Configurations，如图 7-2 所示。

图 7-2: 调试操作



- 在图 7-3 所示的界面上，双击 Compass C Debugger 创建 Compass C 的调试应用。或者右键单击 Compass C Debugger，然后选择 New Configuration 创建调试应用。

图 7-3: Compass C 调试程序加载页面



3. 创建完调试应用后, CompassStudio 会默认加载 Debug 版本的算子进行调试, 当然您也可以单击 Main > C/C++ Application > Browse 选择其他版本的算子进行调试, 如图 7-4 所示。此界面的相关模块参数, 请参考表 7-1。

图 7-4: 创建调试应用

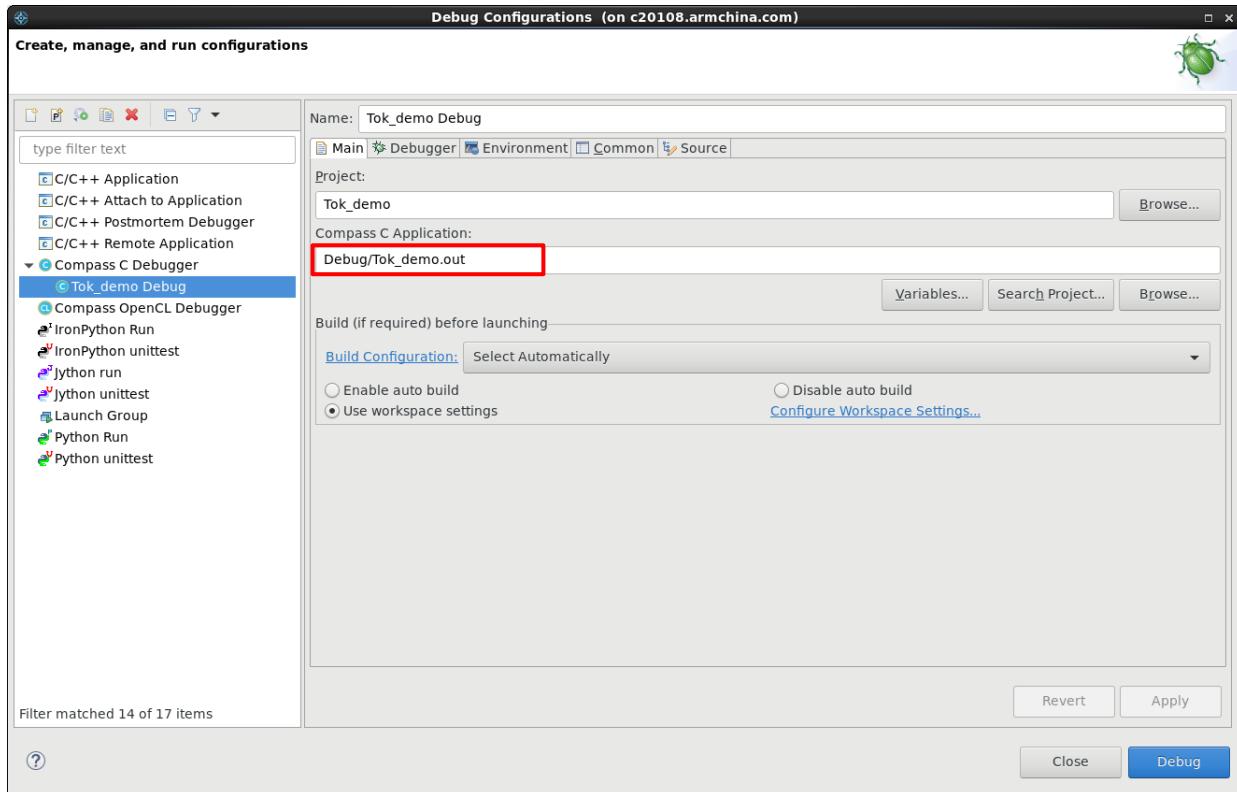
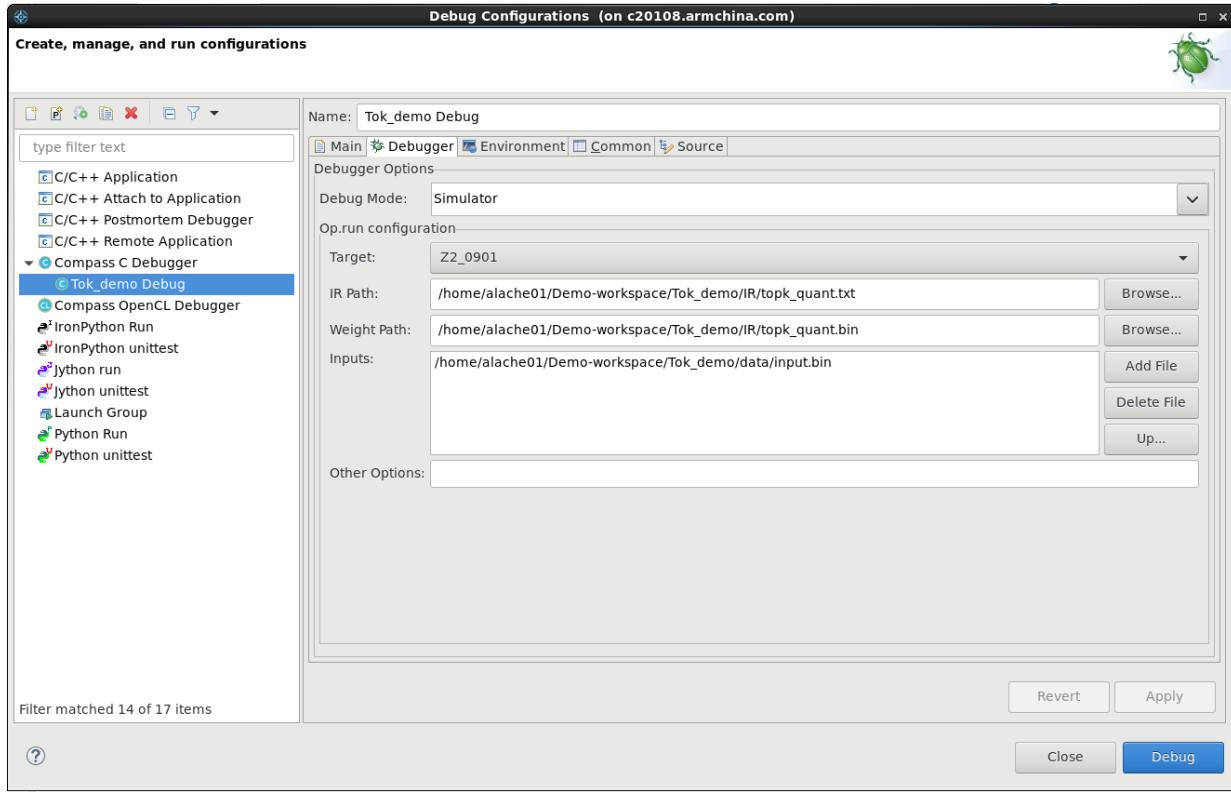


表 7-1: 算子调试模块参数

参数	描述
Main 页签	该页签加载调试工程和调试 elf 文件。CompassStudio 优先加载 Debug 目录下的 elf 文件进行调试。其他配置保持默认即可, 无需修改。
Environment 页签	该页签用来自定义配置调试时的环境。保持默认即可, 无需修改。
Common 页签	配置调试器的相关参数。保持默认即可, 无需修改。
Source 页签	调试时的资源定位器。保持默认即可, 无需修改。
Debugger 页签	用于算子调试的参数配置, 您可以根据需求修改。

4. 单击 Debugger 页签, 选择 Simulator 调试模式。Compass C 算子调试如图 7-5 所示。

图 7-5: Simulator 调试应用



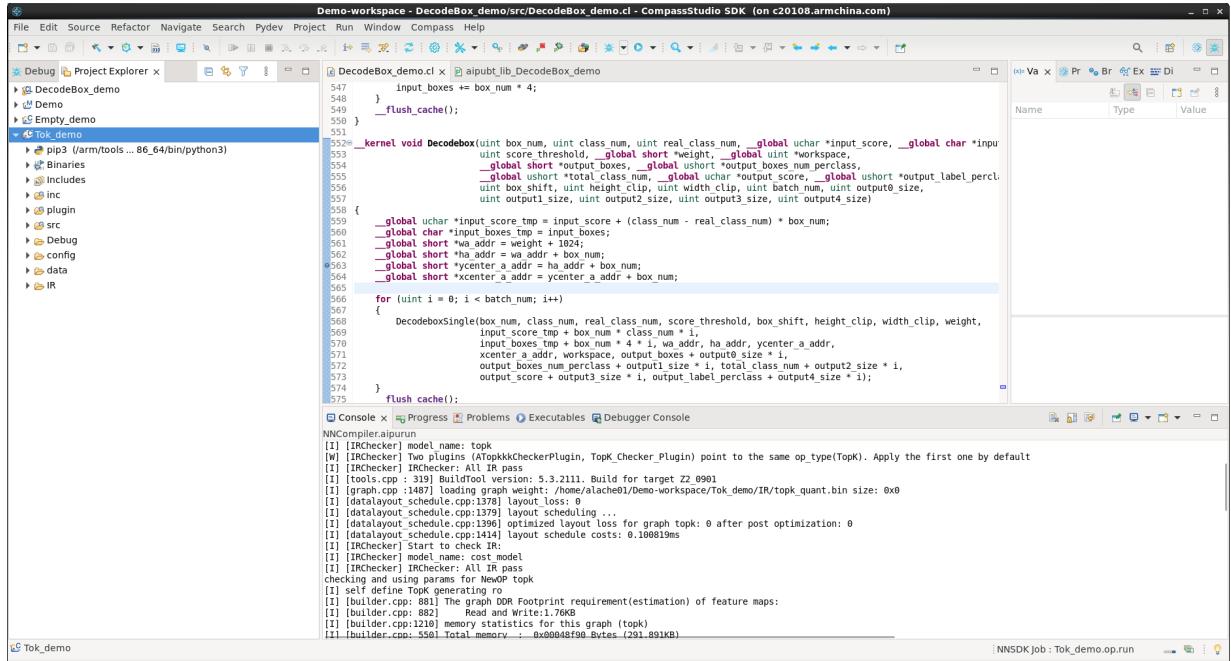
CompassStudio 默认是 Simulator 调试。您也可以在 Debug Mode 下拉列表中选择 Hardware 调试。Simulator 调试页签的参数请参考表 7-2。

表 7-2: Simulator 调试页签的参数

参数	描述
Debug Mode	算子调试模式，分为 Simulator 和 Hardware。
Op.run configuration	算子调试前，需要运行 Compass IR dump 出调试数据。该区域用于配置 Compass IR 相关信息。CompassStudio 内置模板算子，该模块会自动加载。CompassStudio 也会自动加载自定义的算子（请按照 5.1.3 章节的要求自定义）。
IR Path	Op IR 文件。
Weight Path	Op IR 文件。
Inputs	Op IR 输入数据。
Target	Op IR 运行的 target。请确保该模块和 Compiler 的 Target Platform 一致。
Other Options	Op IR 运行的其他参数。

5. 配置完调试信息，单击图 7-5 所示界面上的 Debug 进行调试，调试过程如图 7-6 所示。

图 7-6: Simulator 调试过程

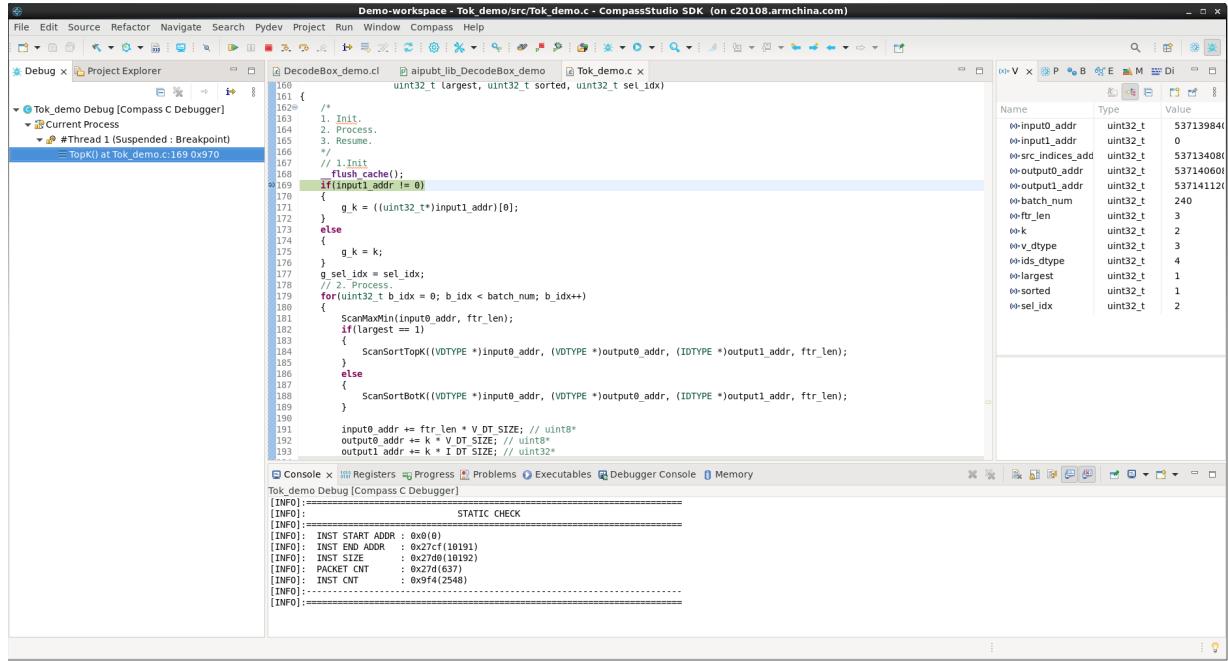


在图 7-6 中，可以观察到，算子调试前，首先运行了 Compass IR dump 出调试需要的数据文件。

6. 查看调试结果。

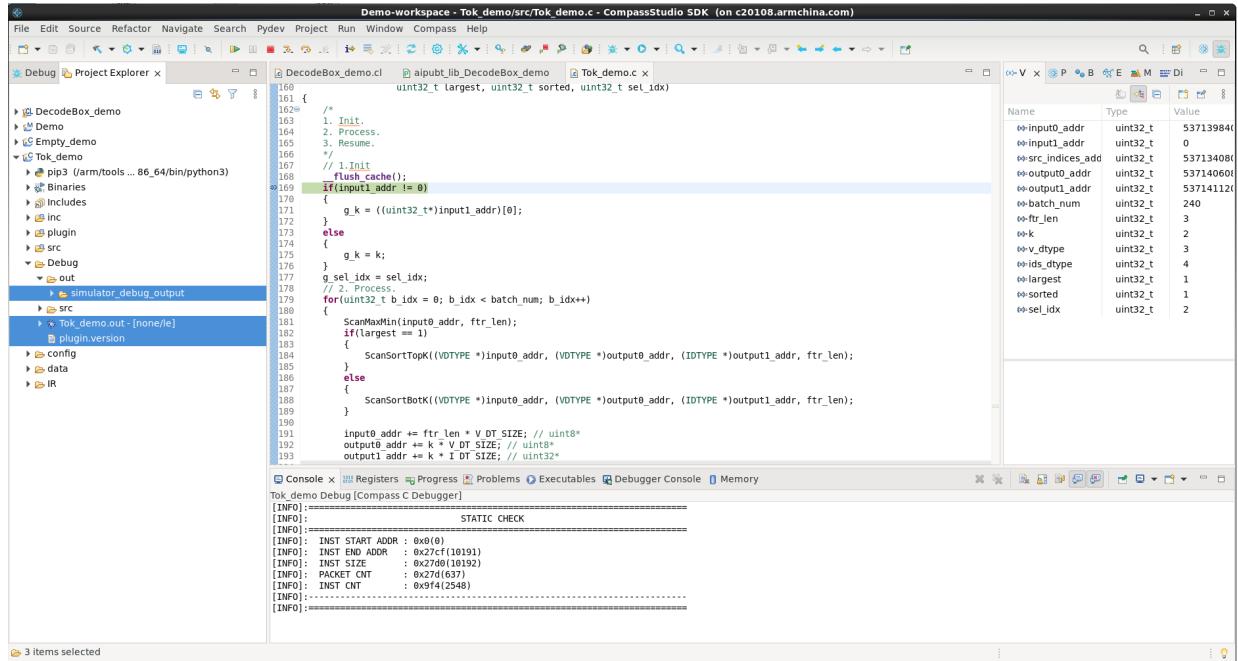
Compass C 算子调试页面如图 7-7 所示，在调试界面，可以执行断点、单步、查看 register、查看 memory 等操作。

图 7-7: Simulator 调试页面



7. 由图 7-7 可知, Simulator 调试已经成功。切换到主界面, 如图 7-8 所示。

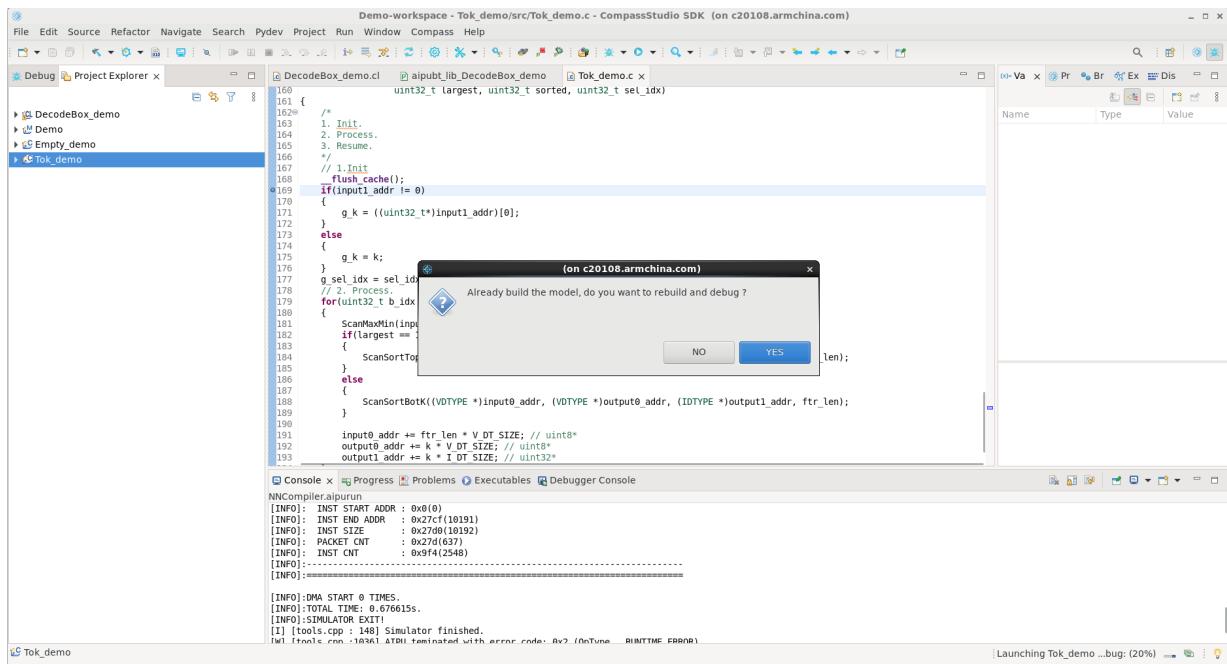
图 7-8: Simulator 调试界面



Simulator 调试成功后，会在 Debug 目录下生成“out/simulator_debug_output”目录，该目录即为 Compass IR dump 出的数据文件目录。还会生成“plugin.version”文件，该文件是对调试时临时部署的算子进行校验，打开该文件进行检查即可。

- 由图 7-8 可知，Simulator 调试成功，会保存 Compass IR dump 出的数据，所以当您再次启动 Debug on simulator（与 Debug on hardware 一致）时，会提示是否需要重新运行 Compass IR。若工程无需修改，不需要重新运行 Compass IR，单击 No 进入 debug。若要修改工程，需要重新运行 Compass IR，单击 Yes 重新运行 Compass IR 并且 dump 出新的数据进行覆盖。Compass IR 运行完成后进入调试，如图 7-9 所示。

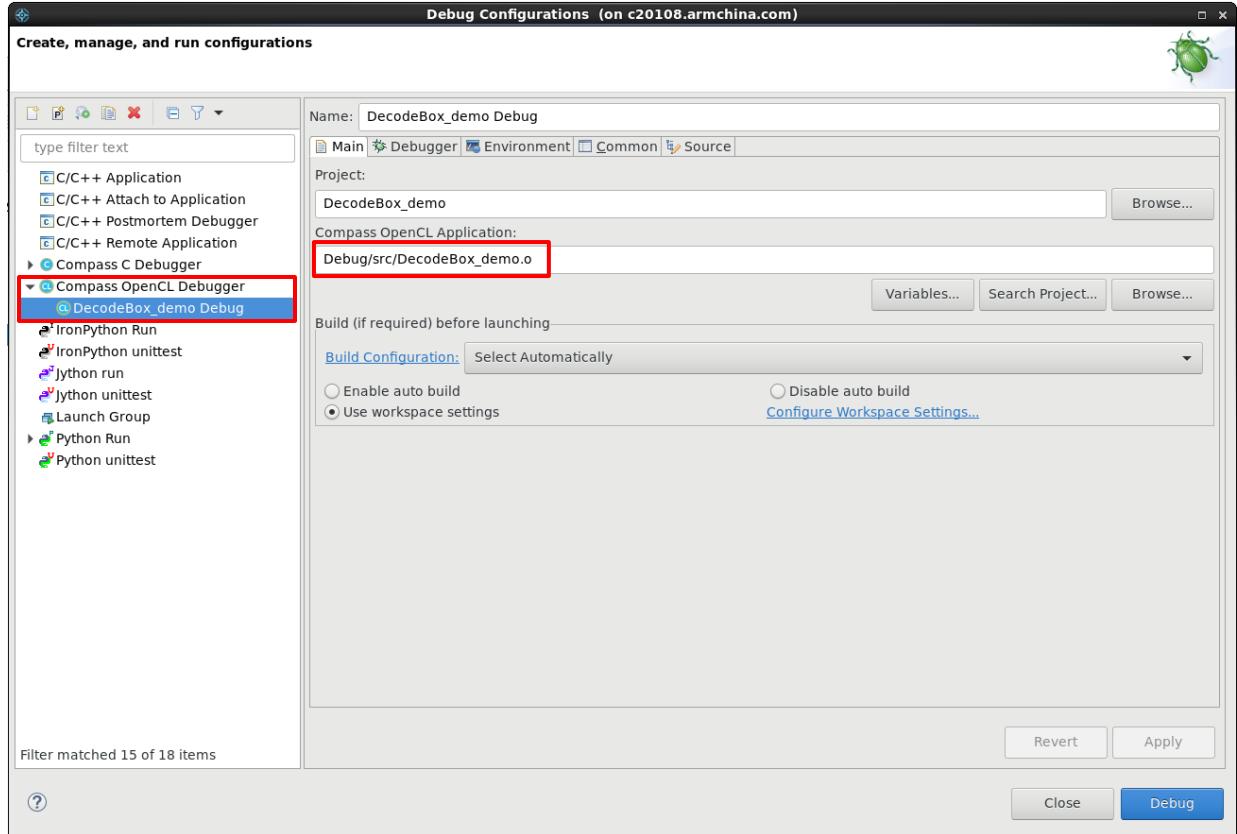
图 7-9：选择是否重新运行 Compass IR



7.1.2 Compass OpenCL Debug on Simulator

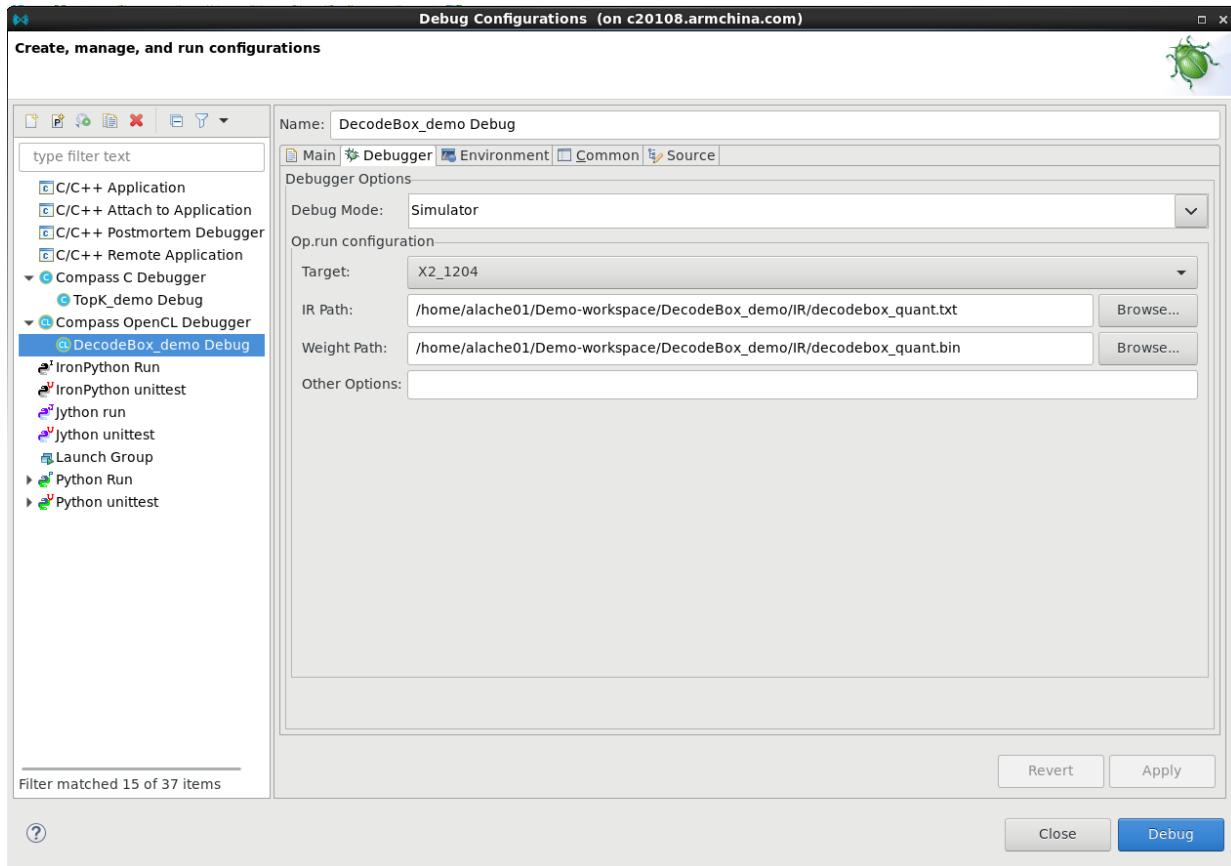
- 右键单击算子 .cl 文件，然后选择 Debug As > Debug Configurations，如图 7-10 所示。

图 7-10: Compass OpenCL 调试程序加载页面



2. 进行 Compass OpenCL 算子调试，如图 7-11 所示。

图 7-11: Compass OpenCL 调试应用



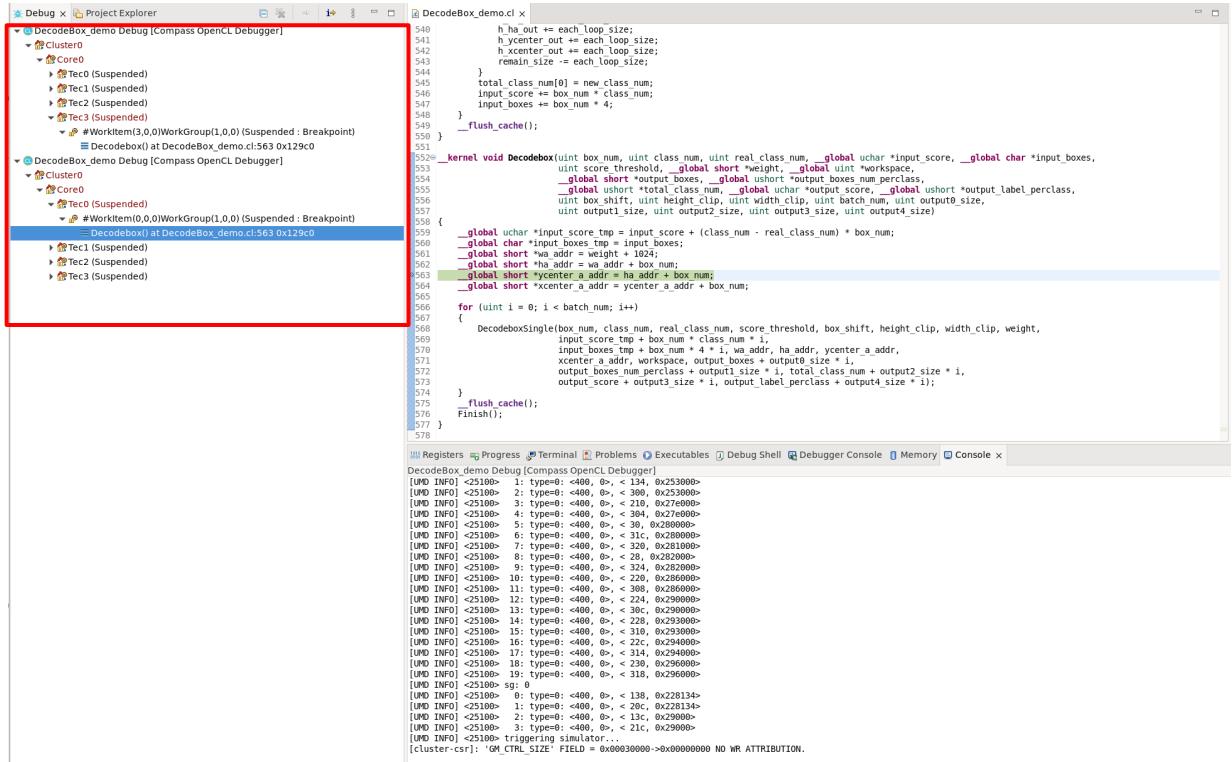
Compass OpenCL 调试步骤和页面参数与 Compass C 调试一致，详细信息可以参考 [7.1.1 Compass C Debug on Simulator](#)。

Compass OpenCL 支持 X2_1204 的单核多 TEC 异步调试和 X2_1204MP3 的多核多 TEC 异步调试。单核多 TEC 异步调试，是指一个 Core 下的 4 个 TEC 都是独立地进行调试，无法进行 4 个 TEC 同时执行同一条调试指令的操作。多核多 TEC 异步调试，是指 1 个 Cluster 下 3 个 Core 都是独立地进行调试，无法进行 3 Core 同时操作同一条调试指令的操作。下面分别进行介绍。

单核多 TEC 异步调试

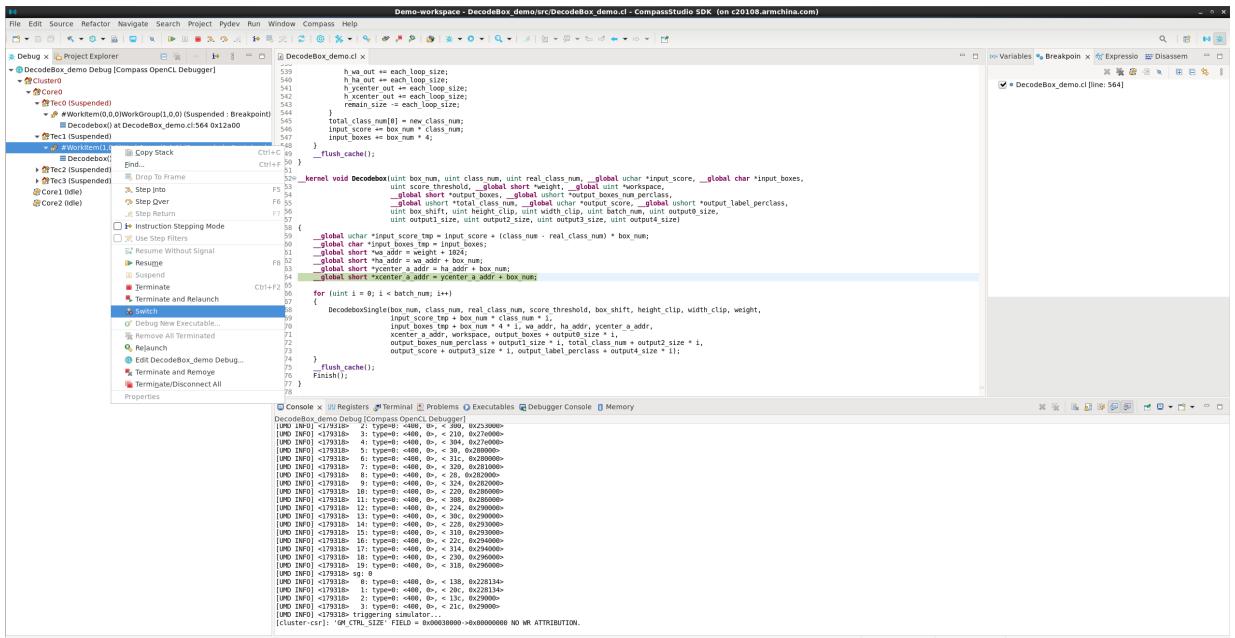
1. 进入调试后，Compass OpenCL 算子调试页面如图 7-12 所示，该调试页面显示 X2_1204 单核多 TEC 信息。

图 7-12: 单核多 TEC 调试页面



- 在图 7-12 所示的界面上，您可以多次创建 Compass OpenCL 的调试程序，可以发现，会随机进入到某一个 TEC。该版本支持在单核下多 TEC 切换的调试。若不进行 TEC 切换的操作，执行单步，跳转等操作则不会切换到其他 TEC 下调试。如果您进行 TEC 切换，则会切换至所需 TEC 下调试，如图 7-13 所示。

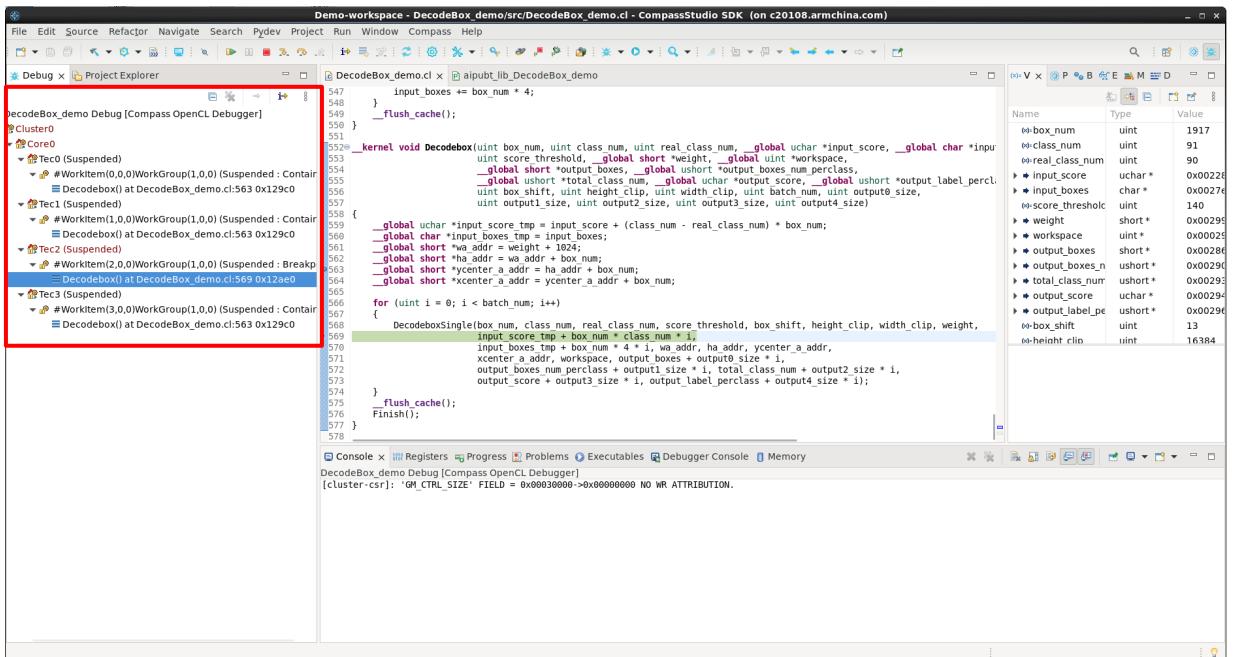
图 7-13: TEC 切换页面



- 右键单击所需切换的 WorkItem，然后选择 Switch，进行 TEC 切换。切换 TEC 后，寄存器、存储器、变量都会自动刷新。

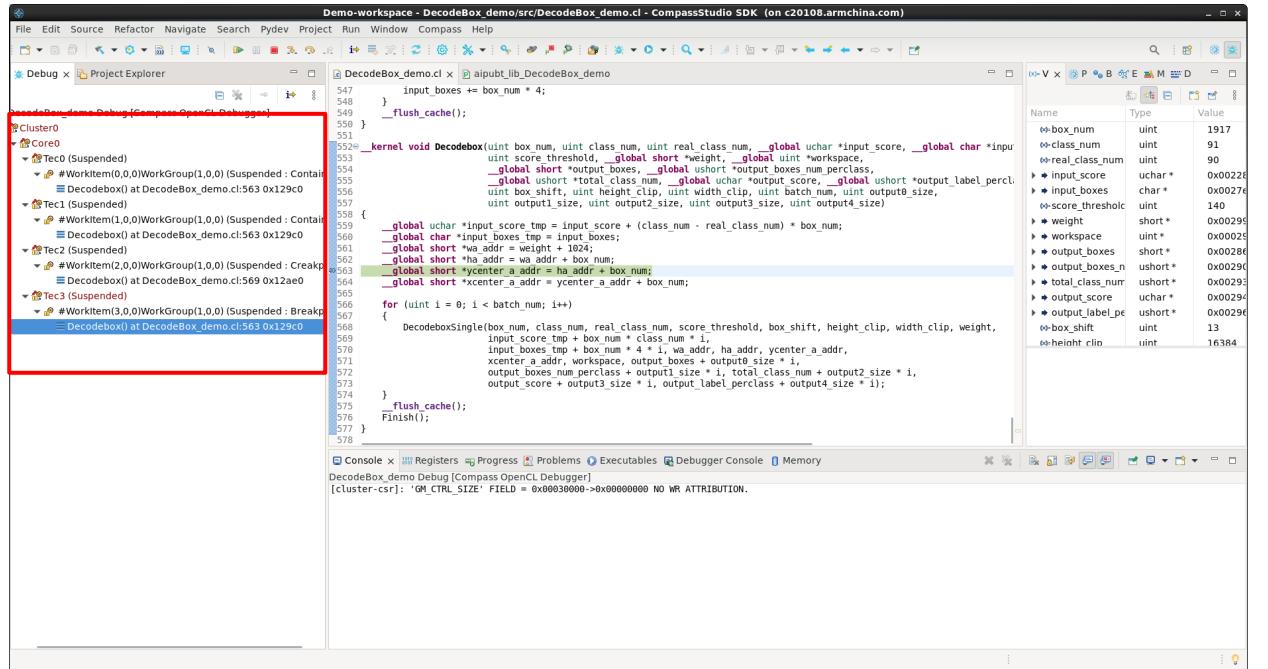
2. 多 TEC 异步调试，如图 7-14 所示。

图 7-14: 多 TEC 异步调试



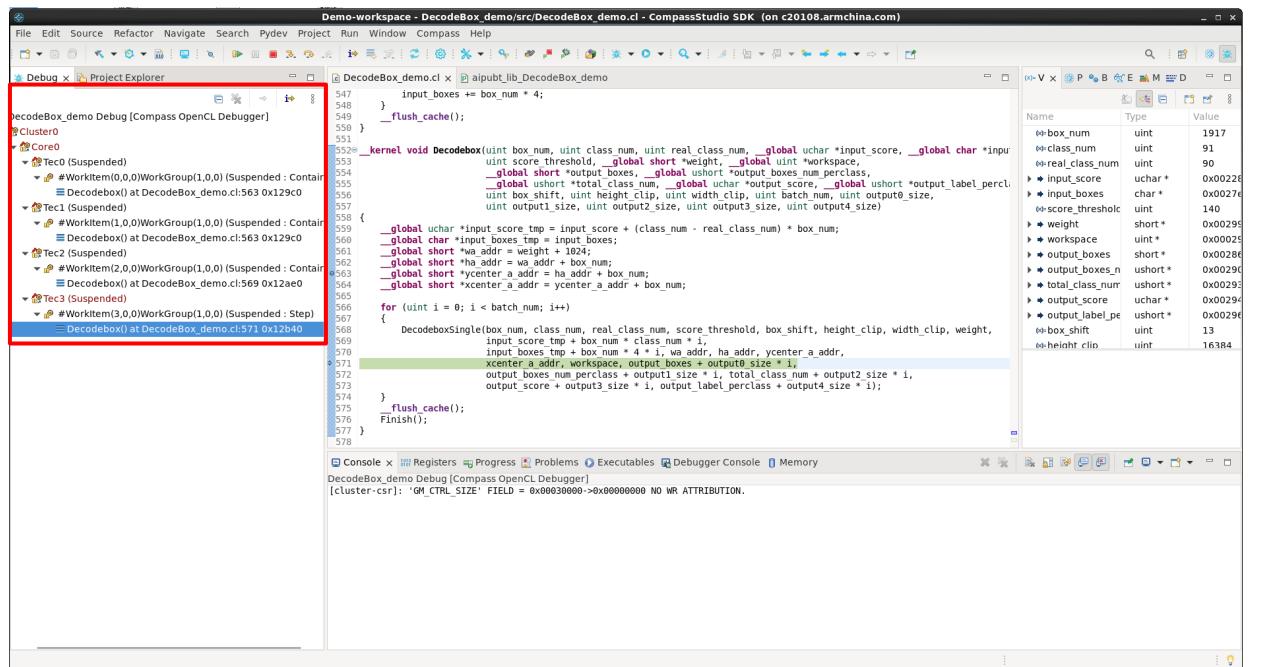
- 在图 7-14 中, 可以发现 Tec2 已经单步到 569 行, 而 Tec0/Tec1/Tec3 依旧停留在 563 行的断点处。选择任一 TEC 进行切换, 如图 7-15 所示, 选择 Tec3 进行切换。

图 7-15: 多 TEC 异步调试



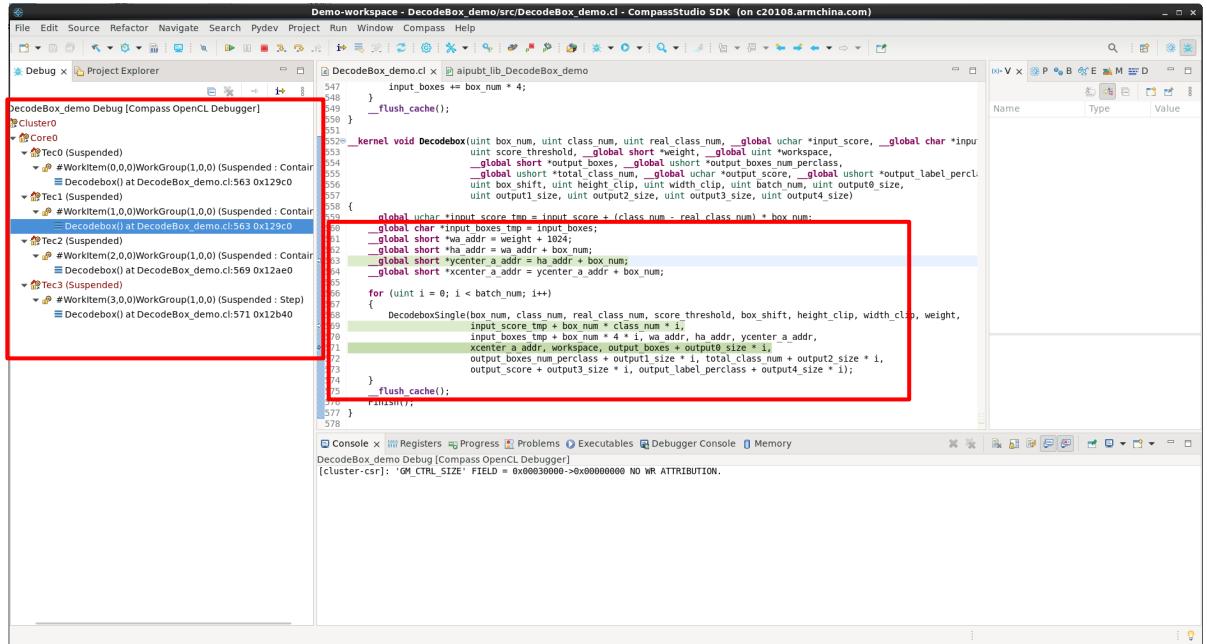
- 如图 7-15 所示, Tec3 切换之后, 即可进行 Tec3 的调试。Tec3 进行单步操作后, 如图 7-16 所示。

图 7-16: 多 TEC 异步调试



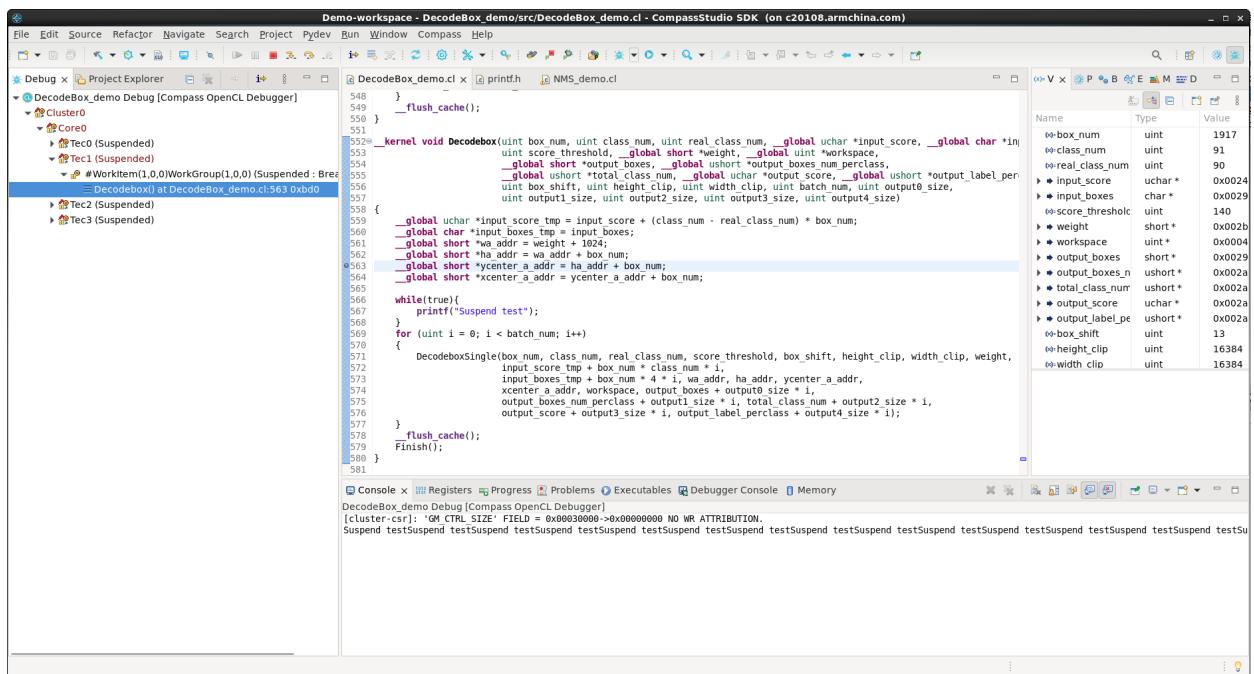
- 如图 7-16 所示, Tec3 进行单步操作后, 停在了 571 行, 而 Tec0/Tec1 由于并没有进行切换操作, 所以依旧停留在 563 行的断点处, Tec2 依旧停留在 569 行。
- 如图 7-17 所示, 单击其他 TEC 可以查看其停在的位置。

图 7-17: 多 TEC 异步调试



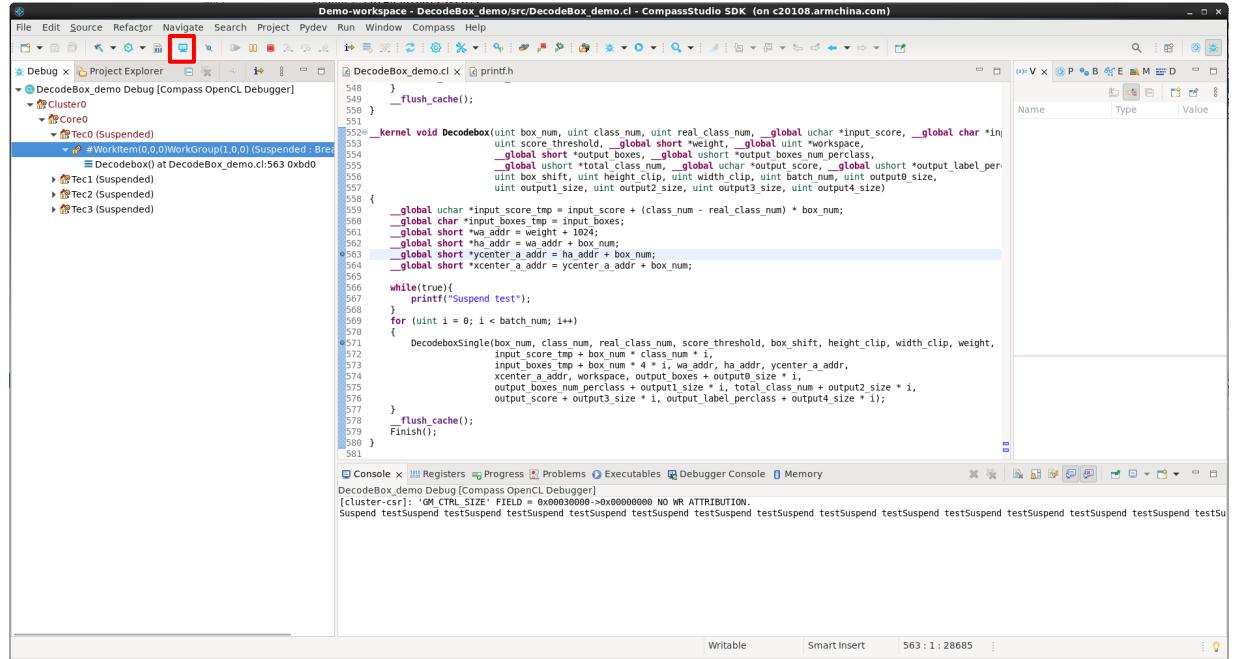
3. 当多 TEC 异步调试进入死循环, 需要执行“Suspend”操作, 如图 7-18 所示。

图 7-18: 死循环下的“Suspend”操作



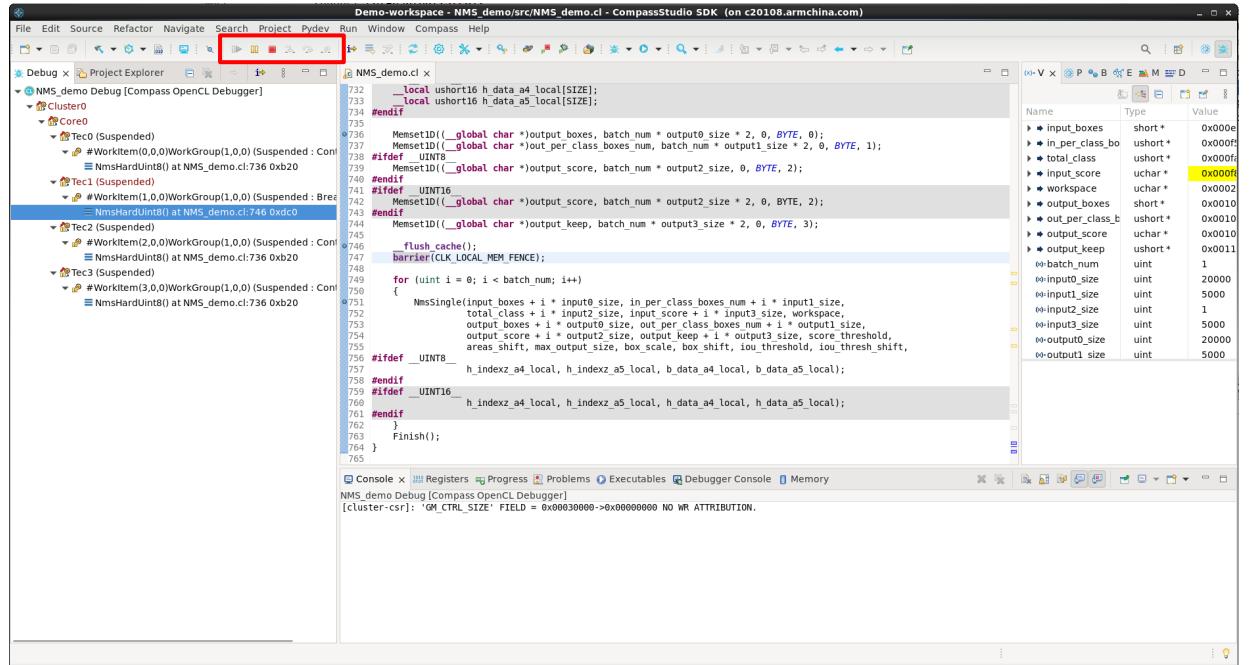
- 在图 7-18 中，在 563 行的断点处执行“Continue”操作，那么程序进入了死循环，这时候就需要执行“Suspend”来中断死循环。在进入死循环后，需要再次选择当前 TEC 下的 WorkItem，如图 7-19 所示。

图 7-19: 死循环下的“Suspend”操作



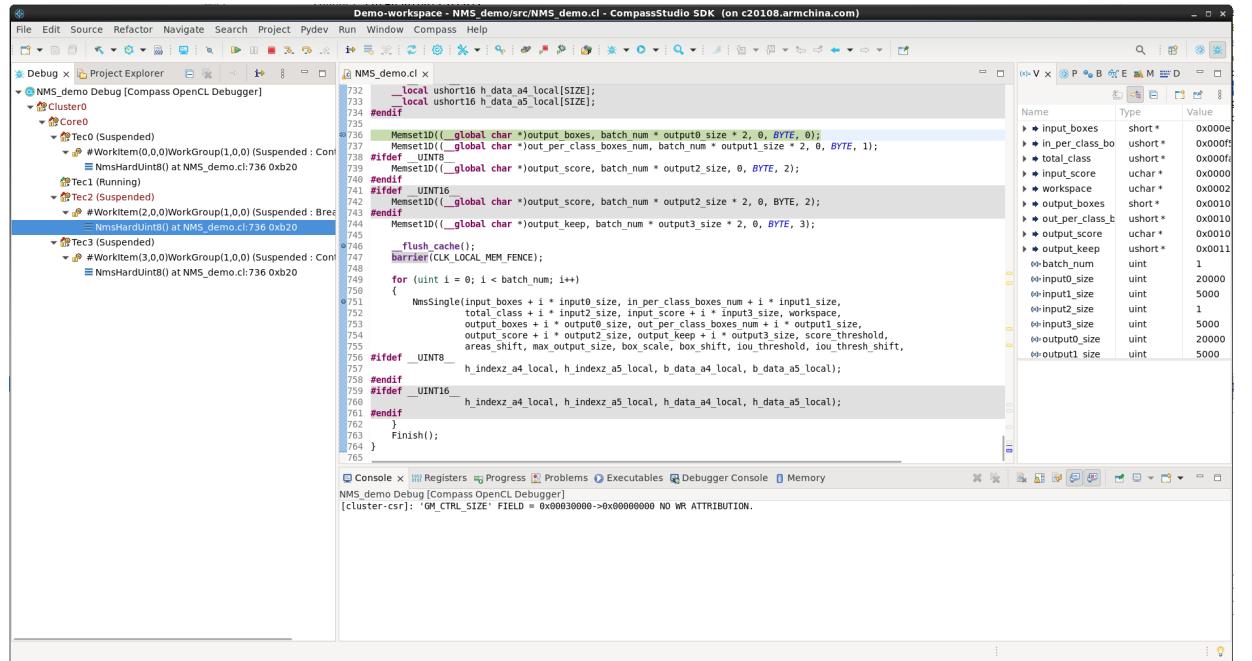
- 对比图 7-18 和图 7-19 后，会发现，进入死循环后，选中当前 WorkItem 时，**Suspend** 图标变成可用状态，点击该图标，即可中断死循环。
4. 多 TEC 异步调试进入“Barrier”等待状态，也需要执行“Suspend”操作，等待其他 TEC 都到达 barrier 后，才可以进行后续的调试操作。
- 该模式下，当前 TEC 进入 barrier 后，进行“Suspend”操作，CompassStudio 会随机切换到任一 TEC 的断点处，此时需要手动操作至 barrier。当所有 TEC 都执行到 barrier 后，才会继续执行剩余代码的调试，如图 7-20 所示。

图 7-20: Barrier 下的“Suspend”操作



- 在图 7-20 中, Tec0 进入 barrier 后, 无法执行其他调试操作 (调试图标变成不可用状态), 单击 Suspend 图标, 如图 7-21 所示。

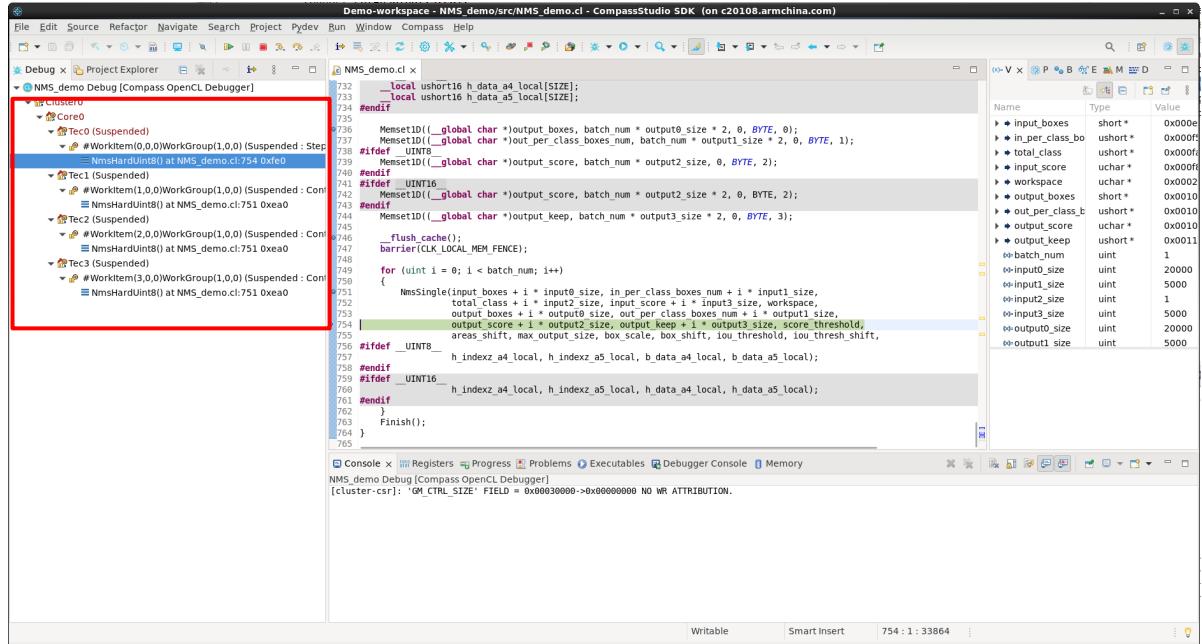
图 7-21: Barrier 下的“Suspend”操作



- 在图 7-21 中, Tec0 会变成“Running”状态, 并且会随机到 Tec2 上。依次执行 Tec3, Tec0 至 barrier。

- 当所有的 TEC 都执行到 barrier 后，会继续在当前 TEC 调试剩余代码，如图 7-22 所示。

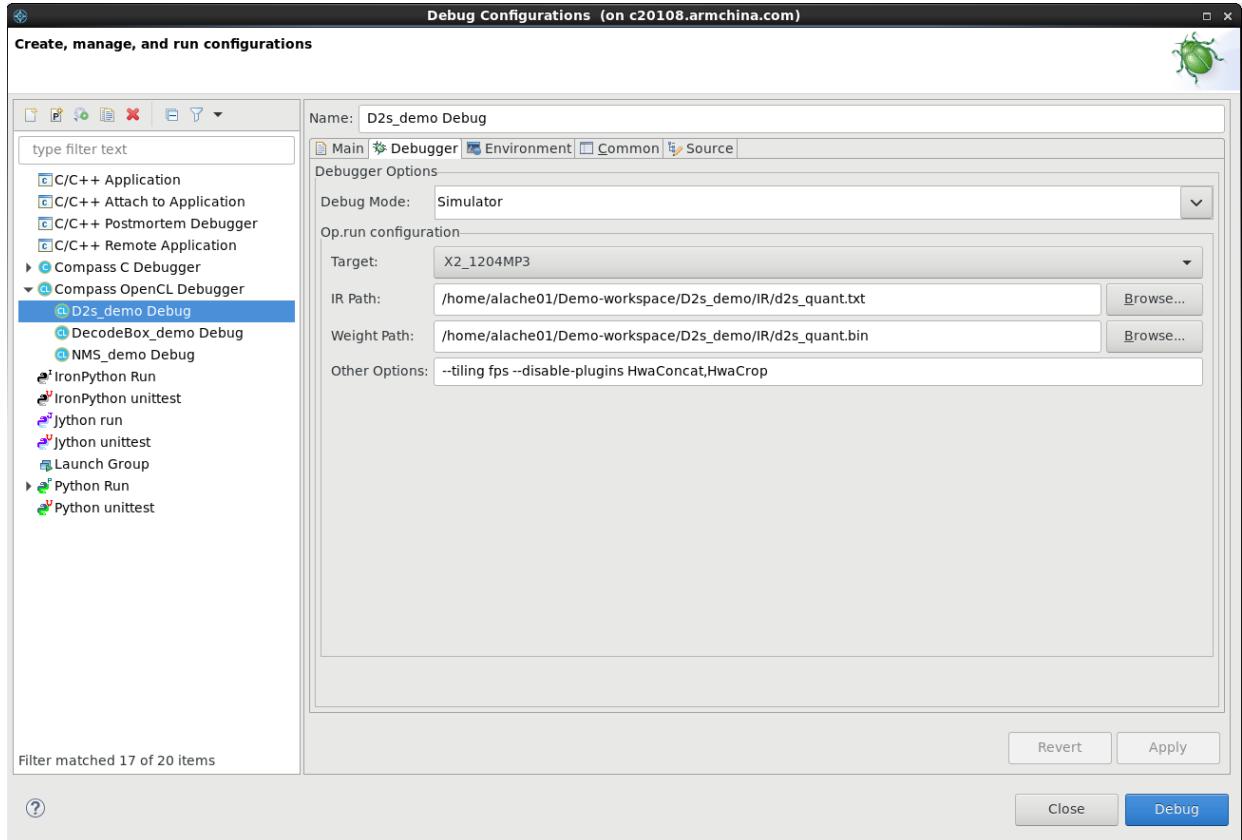
图 7-22: Barrier 下的“Suspend”操作



多核多 TEC 异步调试

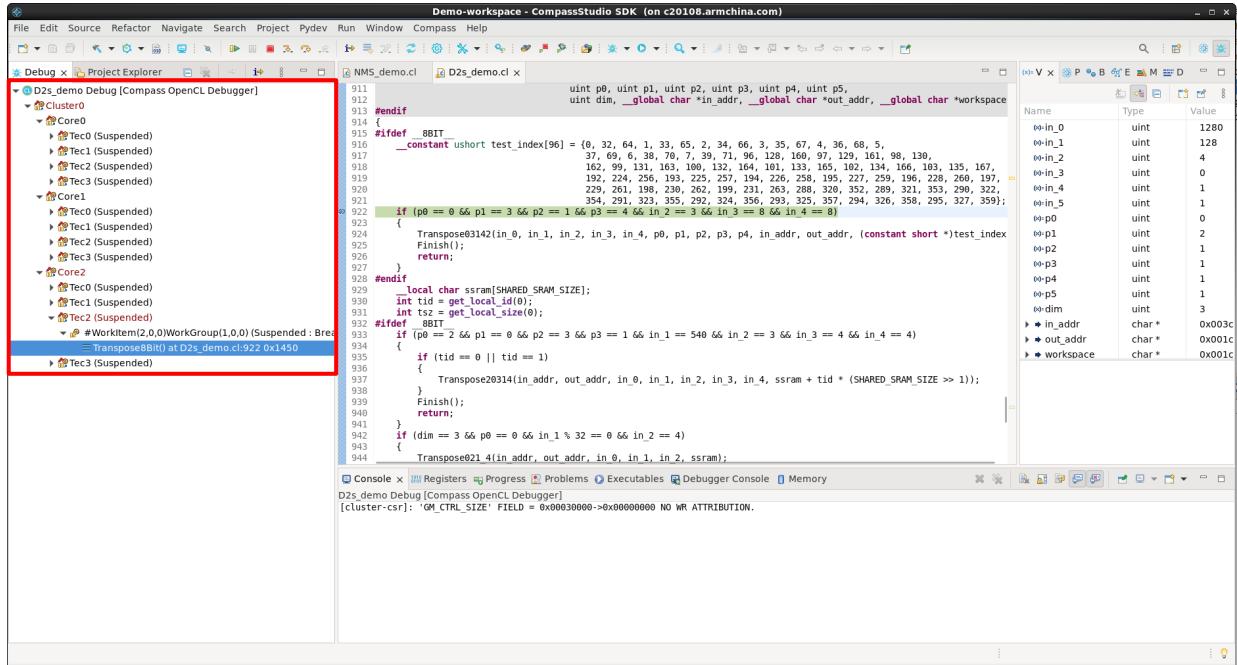
当前版本支持 X2_1204MP3 多核多 TEC 的异步调试，不支持多核多 TEC 的同步调试。在 Target 选择为“X2_1204MP3”的情况下，如果该算子支持“Tiling”，则可以使用多核调试。目前 CompassStudio 集成了支持多核调试的 D2S 算子（其他算子不支持多核调试）。

1. D2S 算子跟 Concat 和 Crop 有冲突，所以需要禁用这两个算子。D2S 调试页面如图 7-23 所示。

图 7-23: D2S 算子调试页面

2. 多核调试页面如图 7-24 所示。

图 7-24: 多核调试页面



CompassStudio 会随机进入某个 Core 进行调试。Core 之间的切换和 TEC 切换操作一样，选择任一 Core 下的任一 TEC，单击 Switch 图标进行切换。请参考[单核多 TEC 异步调试](#)。

3. 程序处于死循环和 Barrier 时，多核多 TEC 调试的处理方法和单核多 TEC 调试一致，请参考[单核多 TEC 异步调试](#)。
4. 未配置 Tiling 时，默认会进入 Core0 调试，剩余两个 Core 则处于“Idle”状态，如图 7-25 和图 7-26 所示。

图 7-25: 不加 Tiling 的调试页面

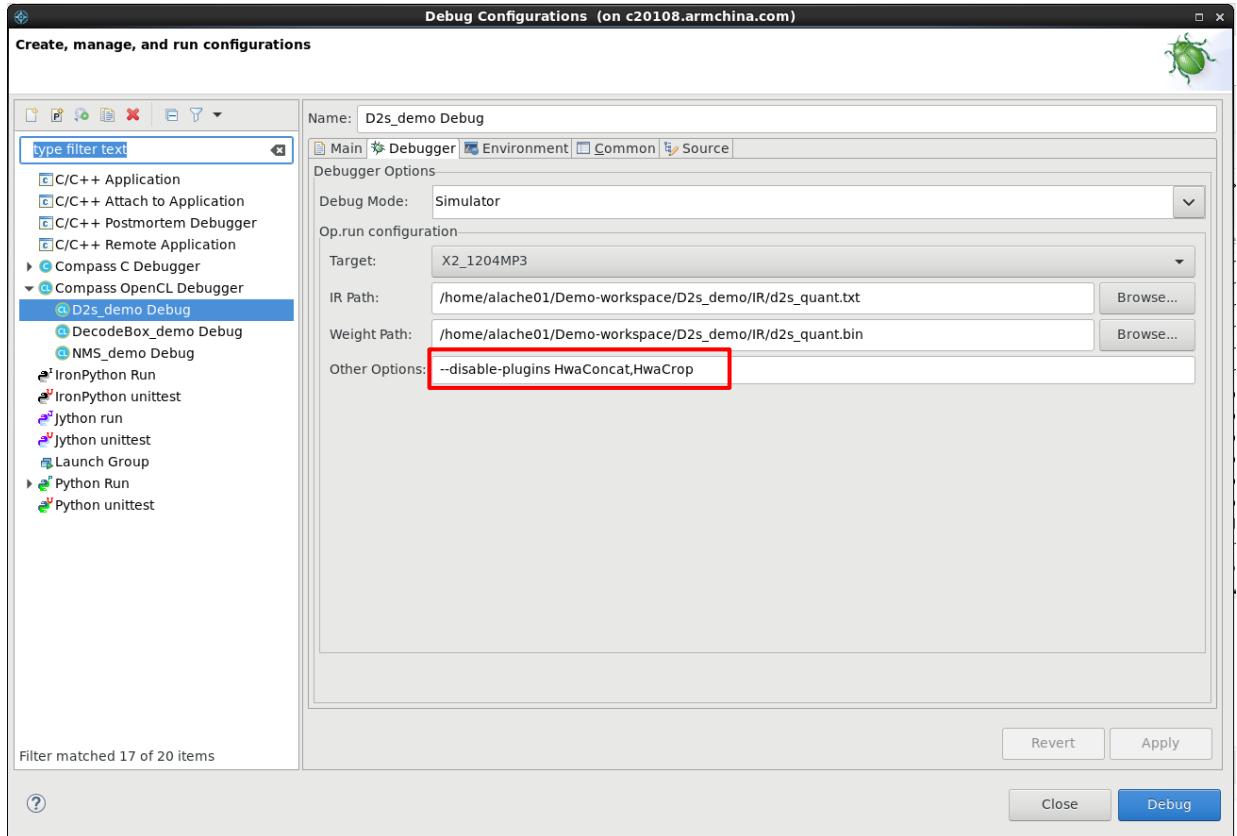
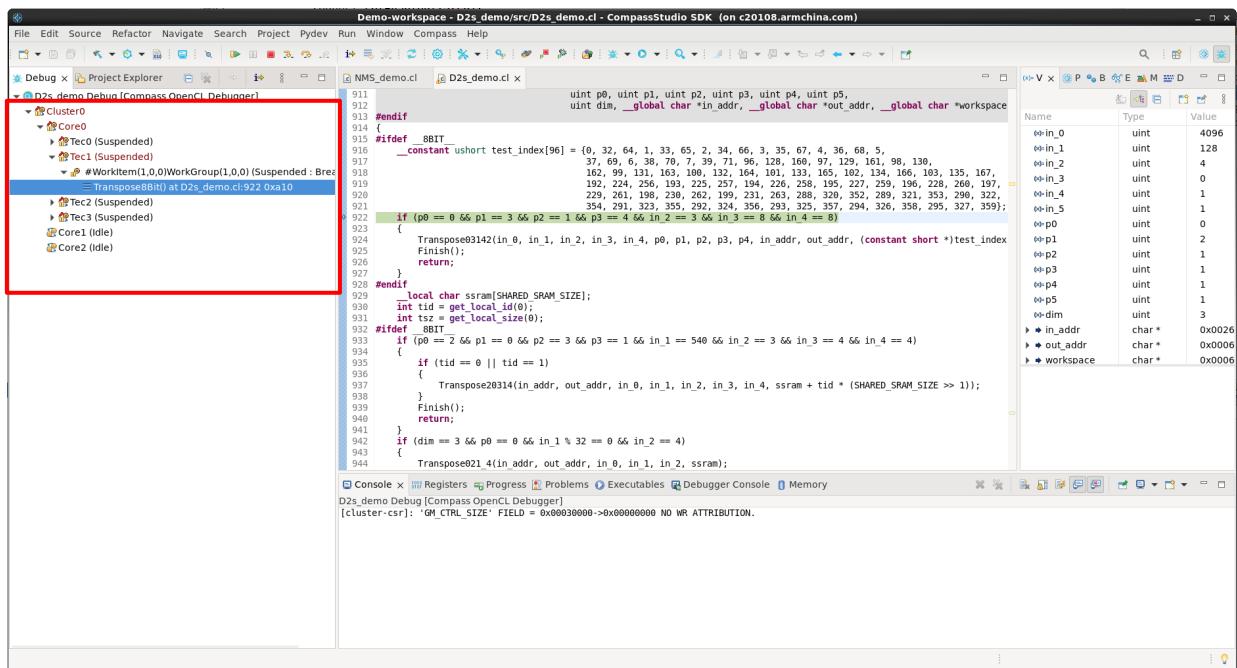


图 7-26: 不加 Tiling 的调试页面

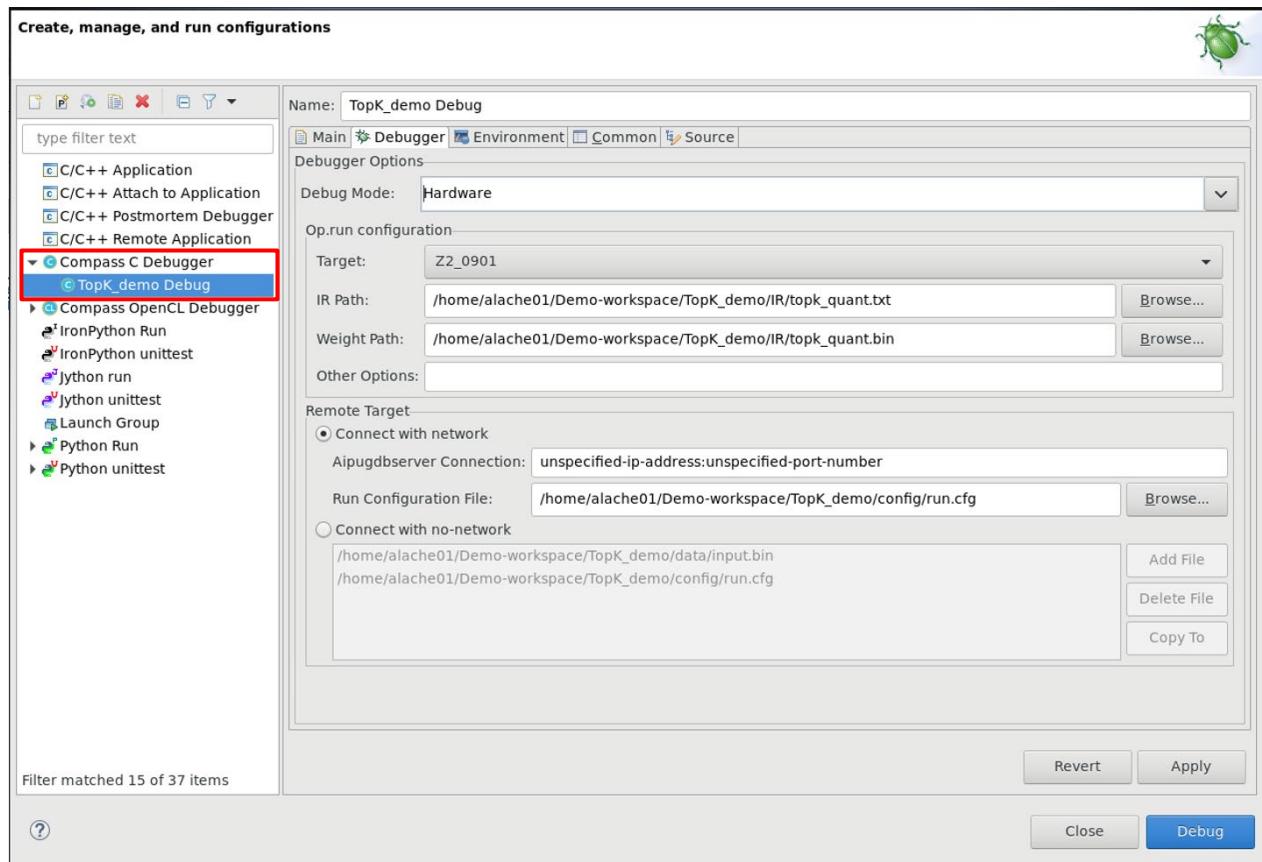


7.2 Debug on Hardware

Compass C 和 Compass OpenCL Debug on Hardware 的步骤一致, Debug on Simulator 和 Debug on Hardware 调试页面一致。本小节以 Compass C Debug on Hardware 为例, 供 Compass OpenCL 参考。

创建“Compass C/OpenCL Debugger”应用, 在 Debug Mode 下拉列表中, 选择 Hardware, 如图 7-27 所示。

图 7-27: Hardware 调试界面



Hardware 调试界面的参数请参考表 7-3。

表 7-3: Hardware 调试界面的参数

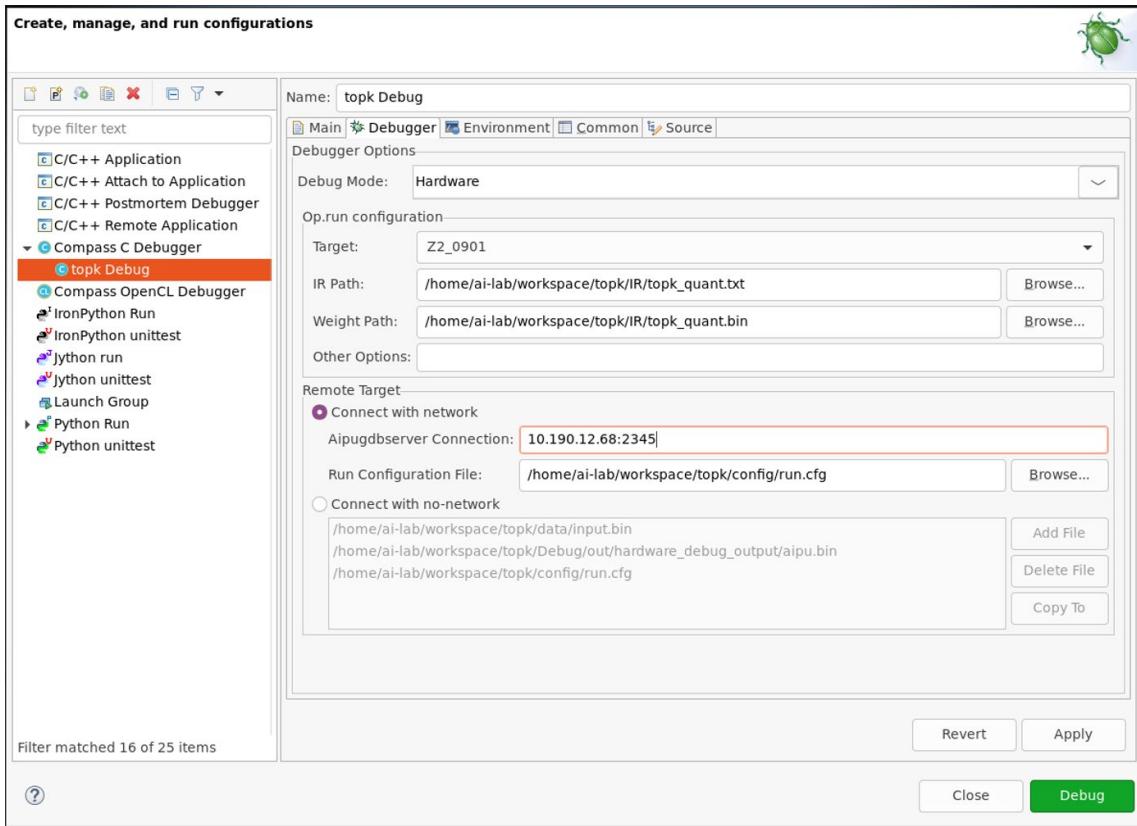
参数	描述
Debug Mode	算子调试模式, 分为 Simulator 和 Hardware。
Op.run configuration	算子调试前, 需要运行 Compass IR dump 出调试数据。该区域用于配置 Compass IR 相关信息。CompassStudio 内置模板算子, 该模块会自动加载。CompassStudio 也会自动加载自定义的算子 (请按照 5.1.3 章节的要求自定义)。

参数	描述
IR Path	Op IR 文件。
Weight Path	Op IR 文件。
Other Options	Op IR 运行的其他参数。
Remote Target	配置开发板数据传输方式, 分为 Connect with network (默认) 和 Connect with no-network。
Connect with network	网线连接开发板, 数据传输以局域网传输。
AipudbgServer Connection	开发板 ip:port, 需要您进行配置。
Run Configuration File	Hardware 调试的配置文件, CompassStudio 会自动加载。
Connect with no-network	无网线连接, 数据靠用户存储设备进行传输。CompassStudio 会自动加载需要传输的数据。

7.2.1 通过网络进行调试

- 在图 7-27 所示的界面上, 在 Remote Target 区域选择 Connect with network, 如图 7-28 所示。

图 7-28: 参数设置



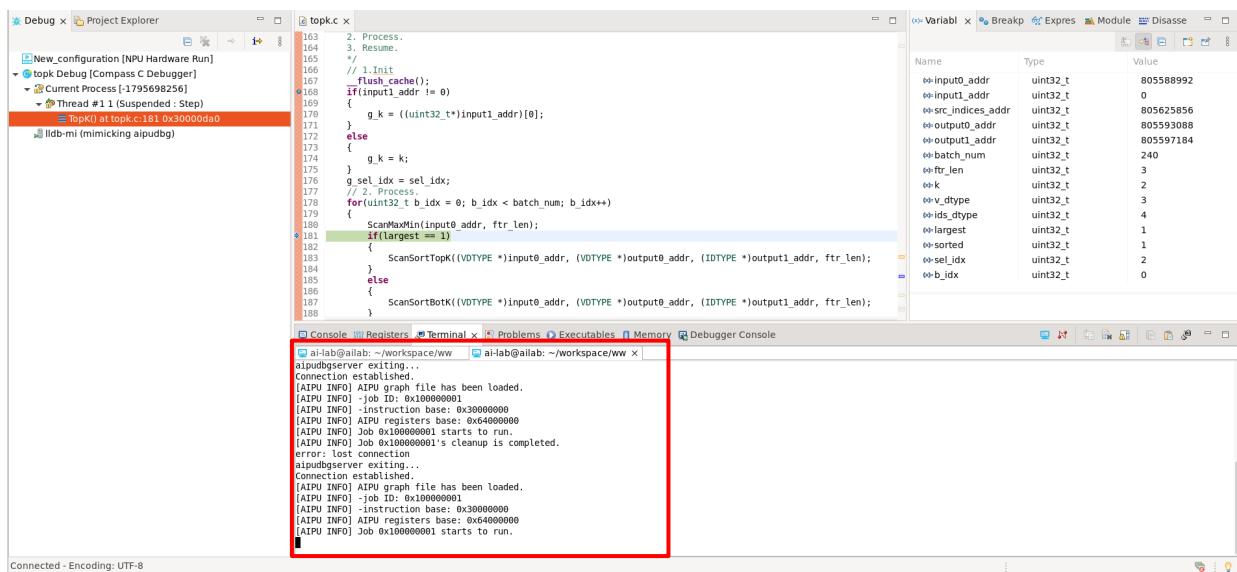
如图 7-28 所示, CompassStudio 会自动填充相关参数。您也可以自定义修改这些参数。

- **AipubgServer Connection:** 配置开发板 IP:Port (port 自定义即可) 。
 - **Run Configuration File:** 配置调试需要的 “run.cfg” 文件。
2. 打开开发板启动 aipubgserver，启动命令为：

```
DEBUGSERVER_INSTALL_PATH/aipubgserver platform --listen *:$port
--server
```

3. 单击 Debug。系统会提示是否配置好开发板的参数以及运行开发板的 aipubgserver，单击 Yes 进入调试界面，如图 7-29 所示。

图 7-29：通过网络进行调试

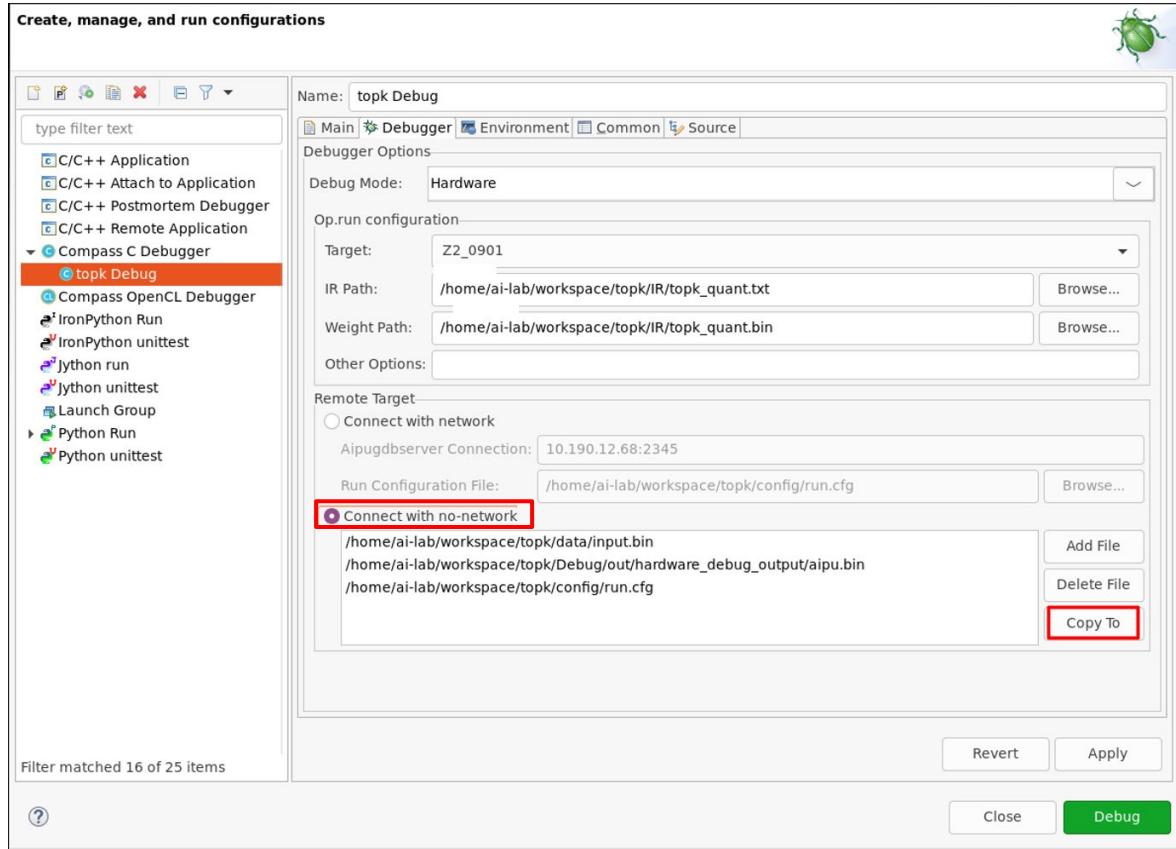


4. 如图 7-29 所示，可以通过 Terminal 打开开发板，监测调试 Job 的相关信息。

7.2.2 不通过网络进行调试

1. 在图 7-28 所示的界面上，在 Remote Target 区域选择 Connect with no-network，CompassStudio 会自动加载需要被传输的数据，如图 7-30 所示。

图 7-30: 不通过网络进行调试



2. 单击 Copy To，选择存储设备进行保存，然后将该数据拷贝至开发板。

3. 打开开发板启动 aipudbgserver，启动命令为：

```
DEBUGSERVER_INSTALL_PATH/aipudbgserver jtag -s ../../run.cfg
```

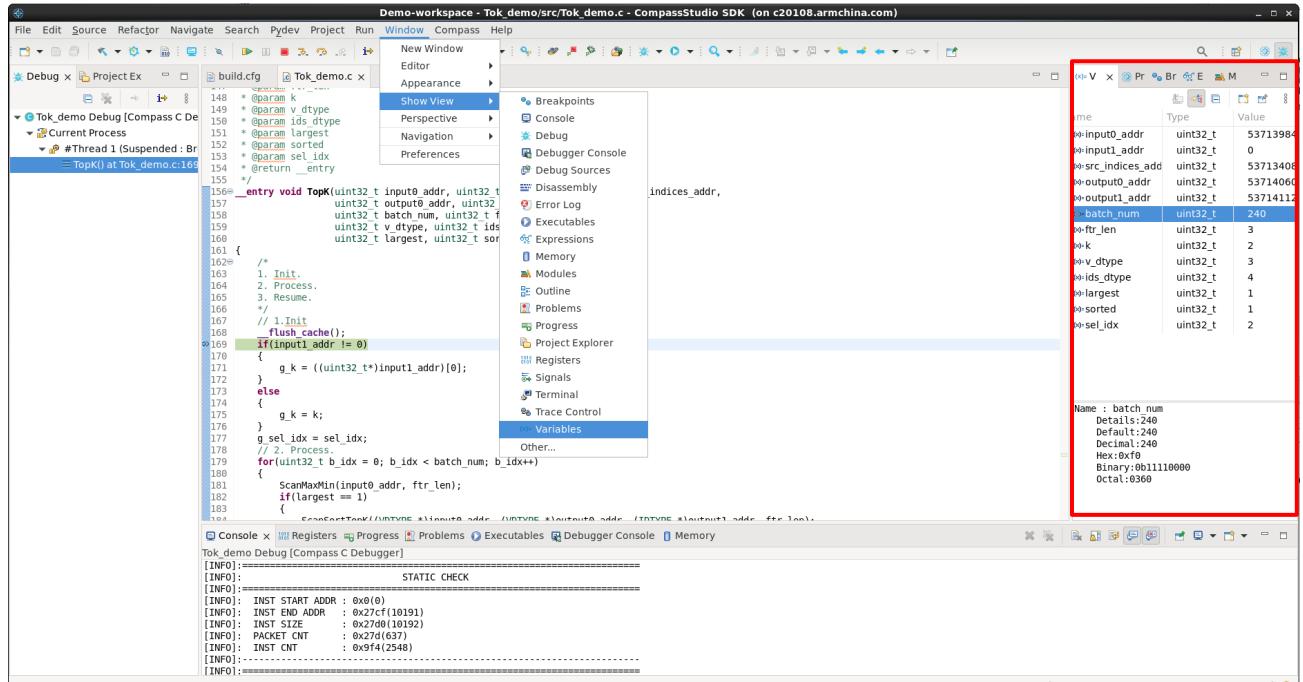
4. 配置“run.cfg”文件，在“run.cfg”中配置步骤 2 拷贝的数据地址。

5. 完成上述操作后，单击 Debug 进行调试。

7.3 查看 Variables

进入调试时，在调试界面，选择 Window > Show View > Variables 打开 Variables 视图，如图 7-31 所示。

图 7-31: 查看 Variables

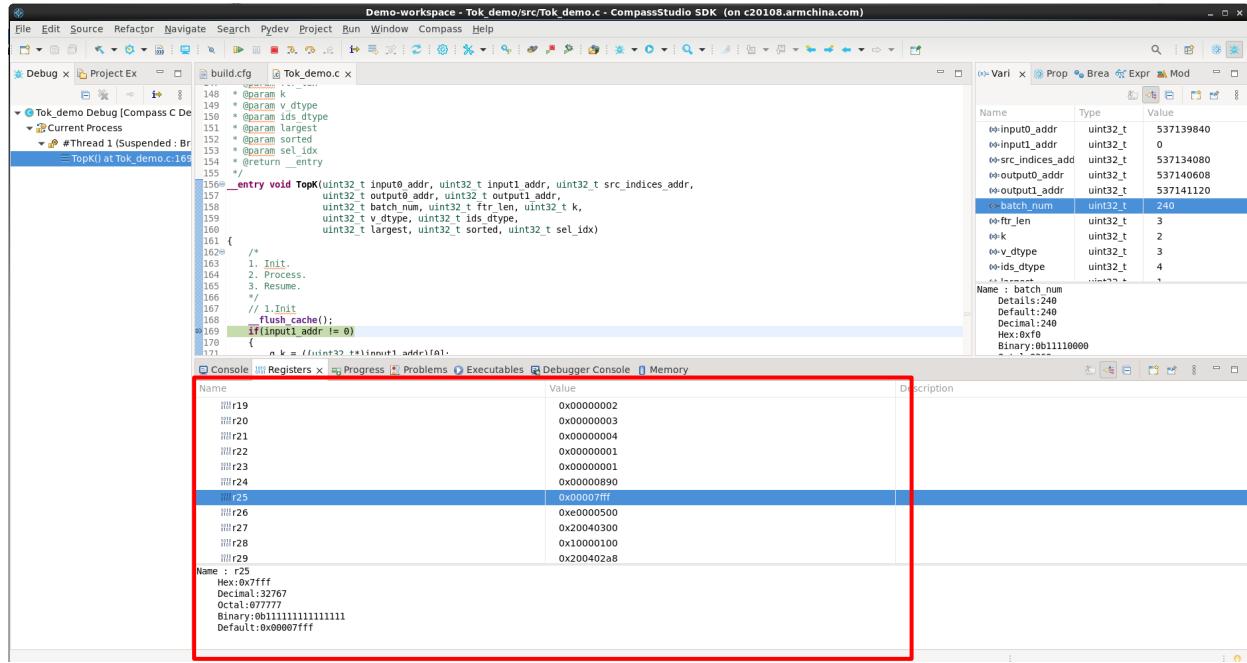


若单步到当前帧的变量，Variables 视图中的变量值会改变。

7.4 查看 Registers

进入调试时，在调试界面，选择 Window > Show View > Registers 打开 Registers 视图，如图 7-32 所示。

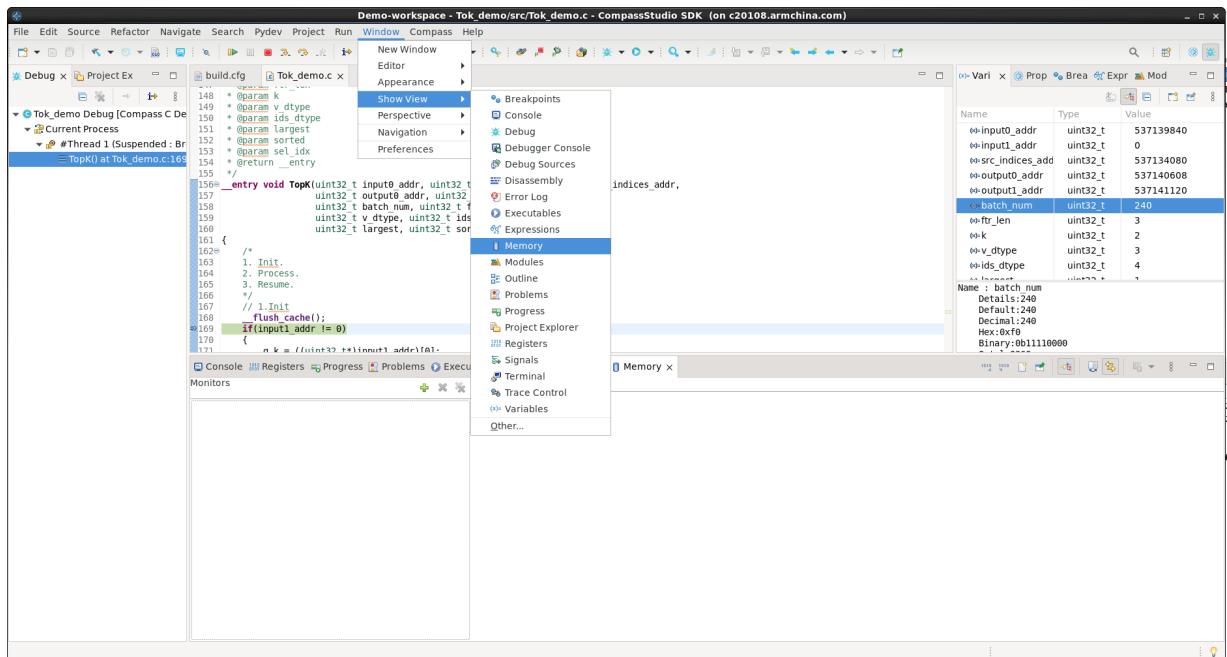
图 7-32: 查看 Registers



7.5 查看 Memory

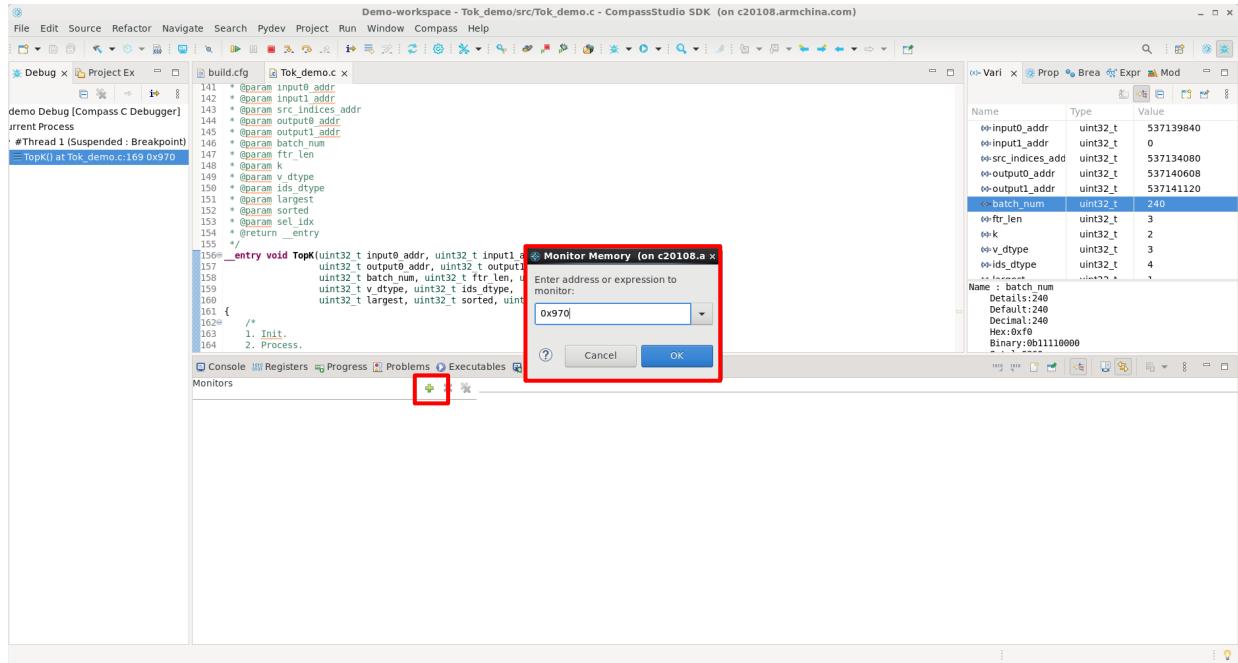
1. 进入调试时，在调试界面，选择 Window > Show View > Memory 打开 Memory 视图，如图 7-33 所示。

图 7-33: 查看 Memory



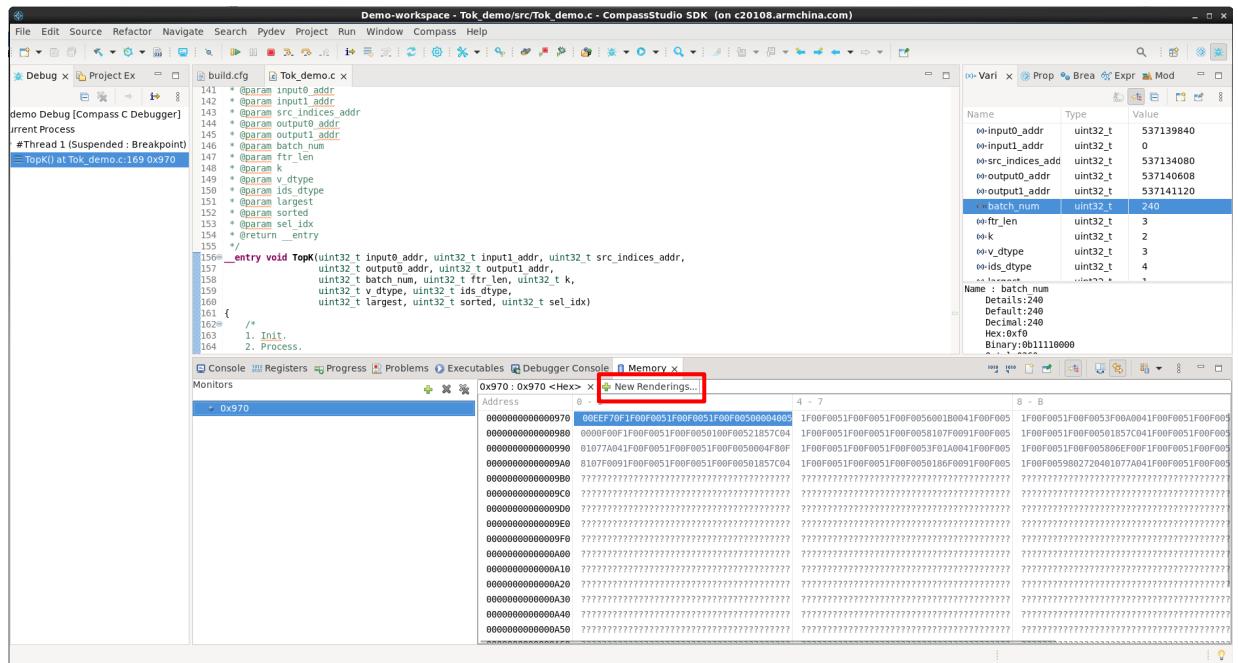
2. 在 Memory 视图, 查看 Memory 地址, 如图 7-34 所示。

图 7-34: 查看 Memory 地址



3. 输入 Memory 地址, 然后单击 OK, 如图 7-35 所示。

图 7-35: 查看 Memory 信息

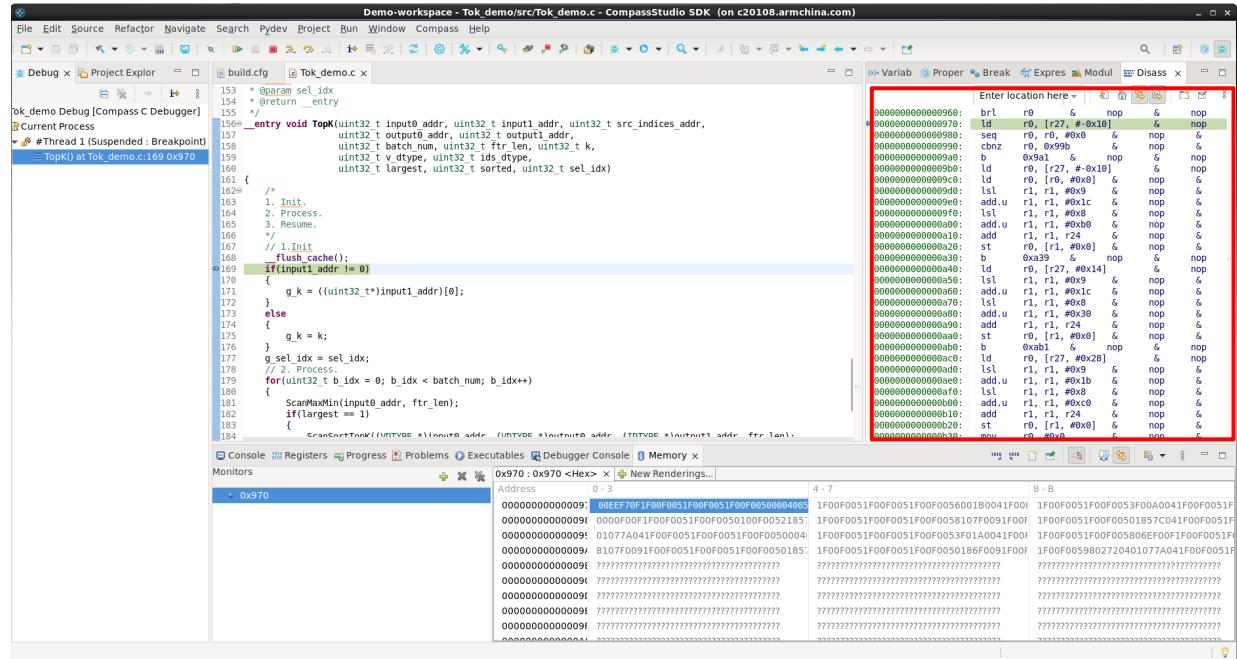


4. 在 Memory 页签, 单击 New Renderings 可以展示其他数据格式。

7.6 查看 Disassembly

进入调试时，在调试界面，选择 Window > Show View > Disassembly 打开 Disassembly 视图，如图 7-36 所示。

图 7-36: 查看 Disassembly



8 其他工具

本章节介绍如何在 CompassStudio 中使用 Python (包括 Python Run/Debug 等), 以及如何使用 Terminal (包括 Local/SSH 等)。

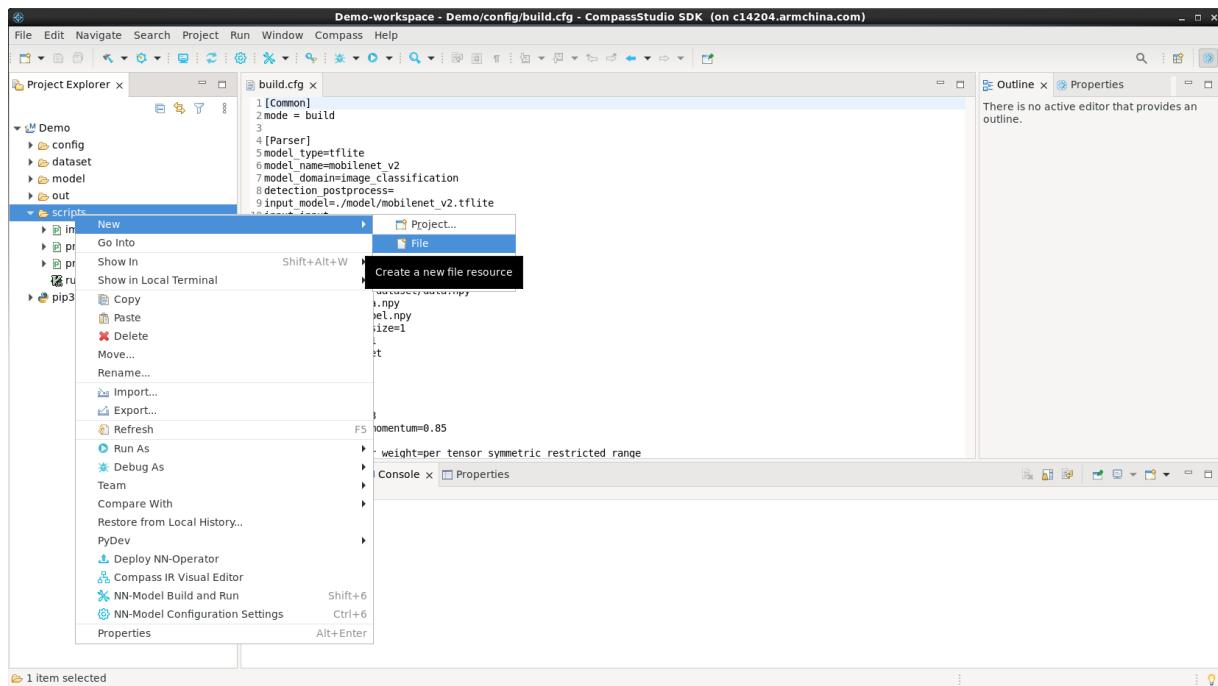
8.1 使用 Python

Python 可在 CompassStudio 中使用, 步骤如下:

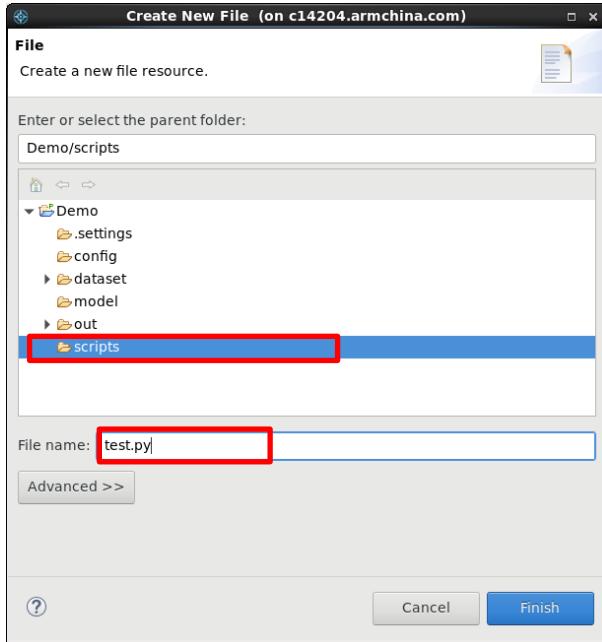
1. 在 Project 中创建 Python 文件。

右键单击需要创建的 Python 文件目录, 然后选择 **New > File**, 如图 8-1 所示。

图 8-1: 创建 Python 文件



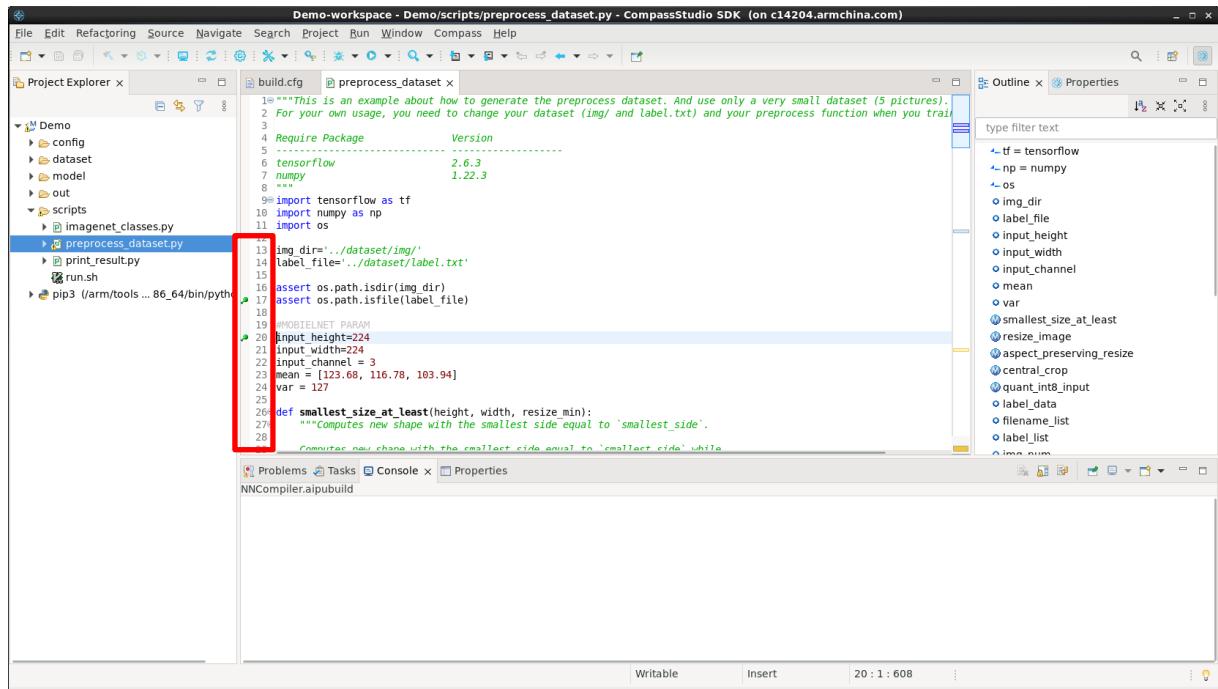
2. 在弹出的界面中创建 Python (.py) 文件, 如图 8-2 所示。

图 8-2: 输入 Python 文件名

3. 单击 **Finish**。完成创建后，在“test.py”中编写 Python 程序。
4. 单击 **Run As > Python Run**，即可运行该脚本。运行日志会显示在 **Console**。
5. 在 Python 脚本中设置断点。

在 Python 脚本需要调试的地方加上断点，双击编辑区左边空白处即可加断点，如图 8-3 所示。

图 8-3: 在 Python 中设置断点



```

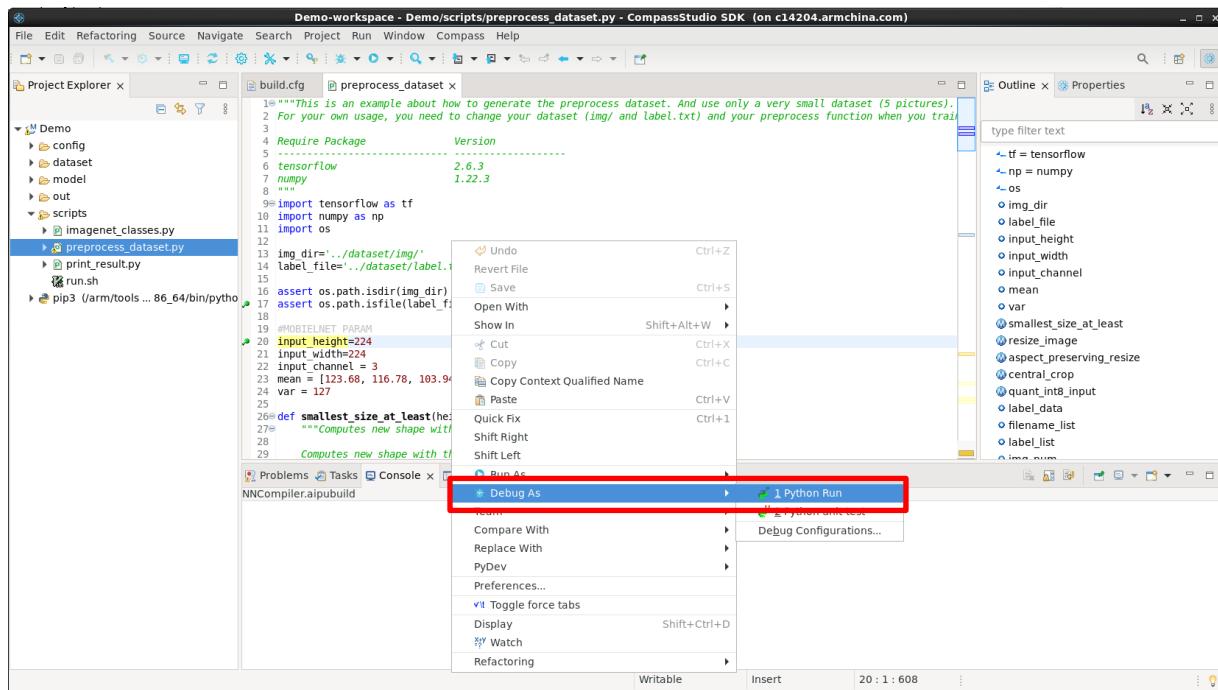
1 # This is an example about how to generate the preprocess dataset. And use only a very small dataset (5 pictures).
2 # For your own usage, you need to change your dataset (img/ and label.txt) and your preprocess function when you train.
3
4 Require Package Version
5 -----
6 tensorflow 2.6.3
7 numpy 1.22.3
8 ***
9 import tensorflow as tf
10 import numpy as np
11 import os
12
13 img_dir='./dataset/img/'
14 label_file='./dataset/label.txt'
15
16 assert os.path.isdir(img_dir)
17 assert os.path.isfile(label_file)
18
19 #MOBILENET PARAM
20 input_height=224
21 input_width=224
22 input_channel = 3
23 mean = [123.68, 116.78, 103.94]
24 var = 127
25
26 def smallest_size_at_least(height, width, resize_min):
27     """Computes new shape with the smallest side equal to 'smallest_side'.
28     Computes new shape with the smallest side equal to 'smallest_side', while

```

6. 进行 Python 调试。

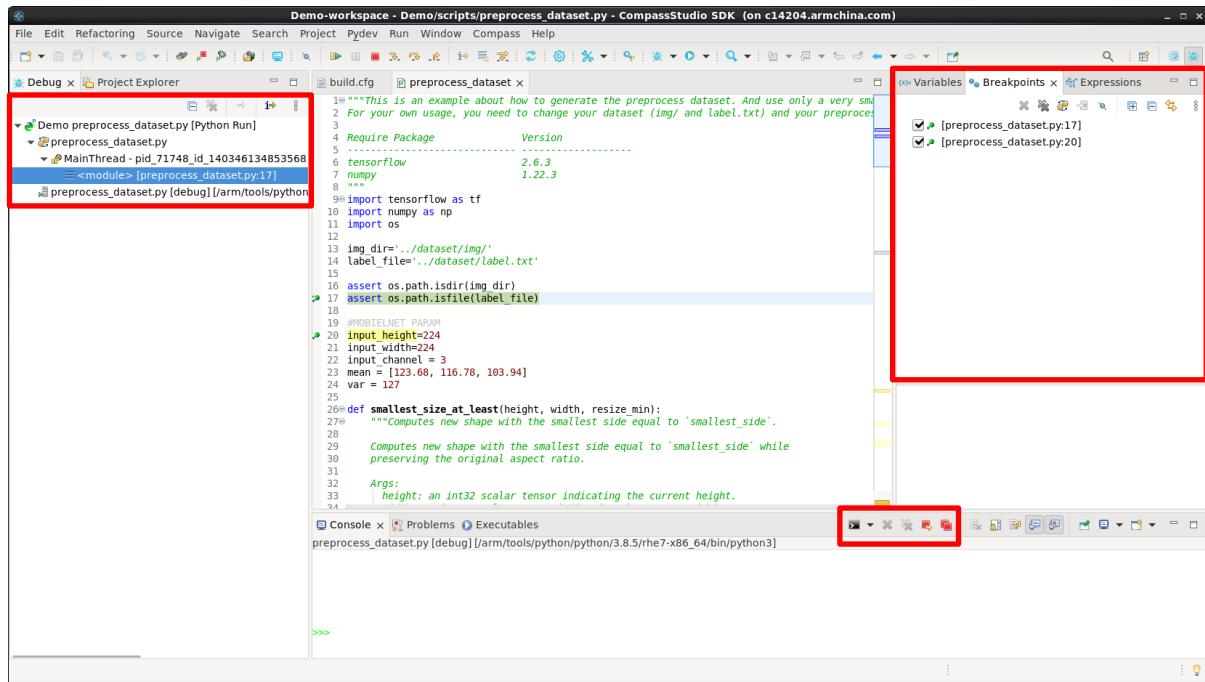
选择 Debug As > Python Run, 如图 8-4 所示。

图 8-4: Python 调试



Python 调试界面, 如图 8-5 所示。

图 8-5: Python 调试界面



CompassStudio 中 Python 的调试步骤和其他语言调试方式一样, 在工具栏里面单击 step into/step over/step return/resume/suspend/terminate 等。具体操作, 进入调试状态后, 单击工具栏对应图标即可。

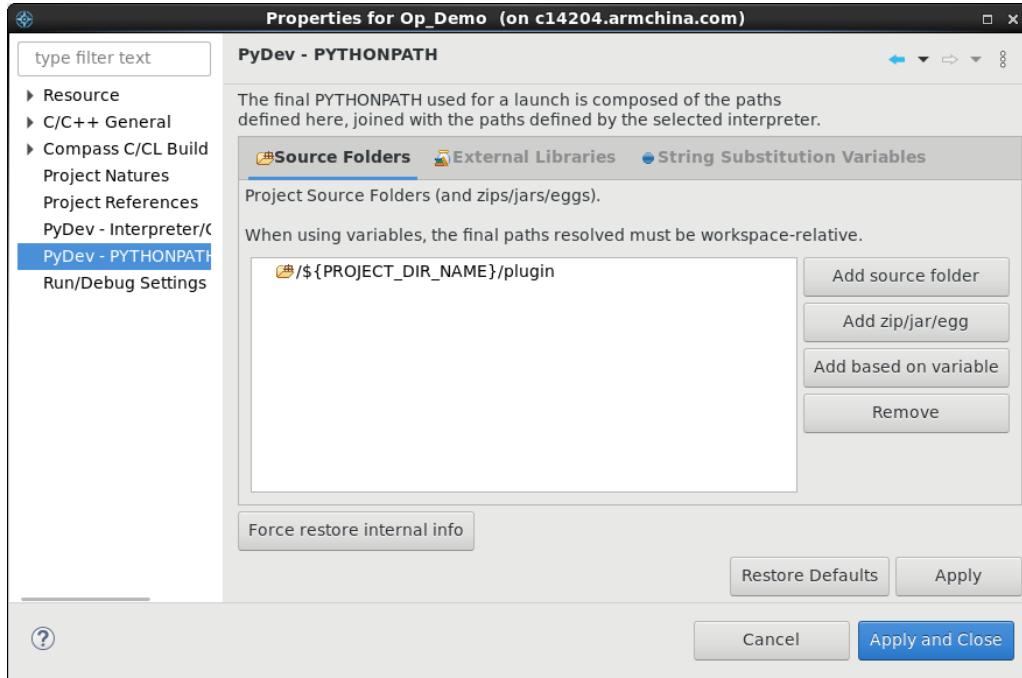
如图 8-5 所示, 在右上角的 Breakpoints 视图中, 您可以右键单击一个断点, 然后选择 Enable 或 Disable 启用或禁用该断点, 也可以勾选该断点。也可以使用 Ctrl + 鼠标左键或者 Shift + 鼠标左键选择多个断点进行操作。

8.2 在算子工程中增加 Python 语法

CompassStudio 默认对算子工程根目录下的 plugin 文件夹提供 Python 语法支持, 其他目录下不支持 Python 语法。要想增加 CompassStudio 对其他目录的 Python 语法支持, 请参考如下步骤:

1. 右键单击算子工程, 然后选择 Properties > PyDev-PYTHONPATH。CompassStudio 默认配置如图 8-6 所示。

图 8-6: 自定义算子工程 Python 语法解析



2. 单击 **Add source folder**, 将需要支持 Python 语法的目录添加到工程属性中。

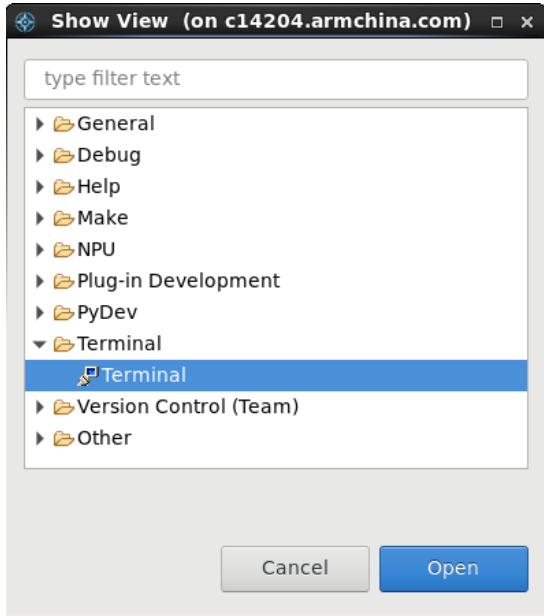


请勿将算子工程根目录添加至上图所示的工程属性中，否则会使整个算子 C 工程变成 Python 工程。

8.3 使用 Terminal

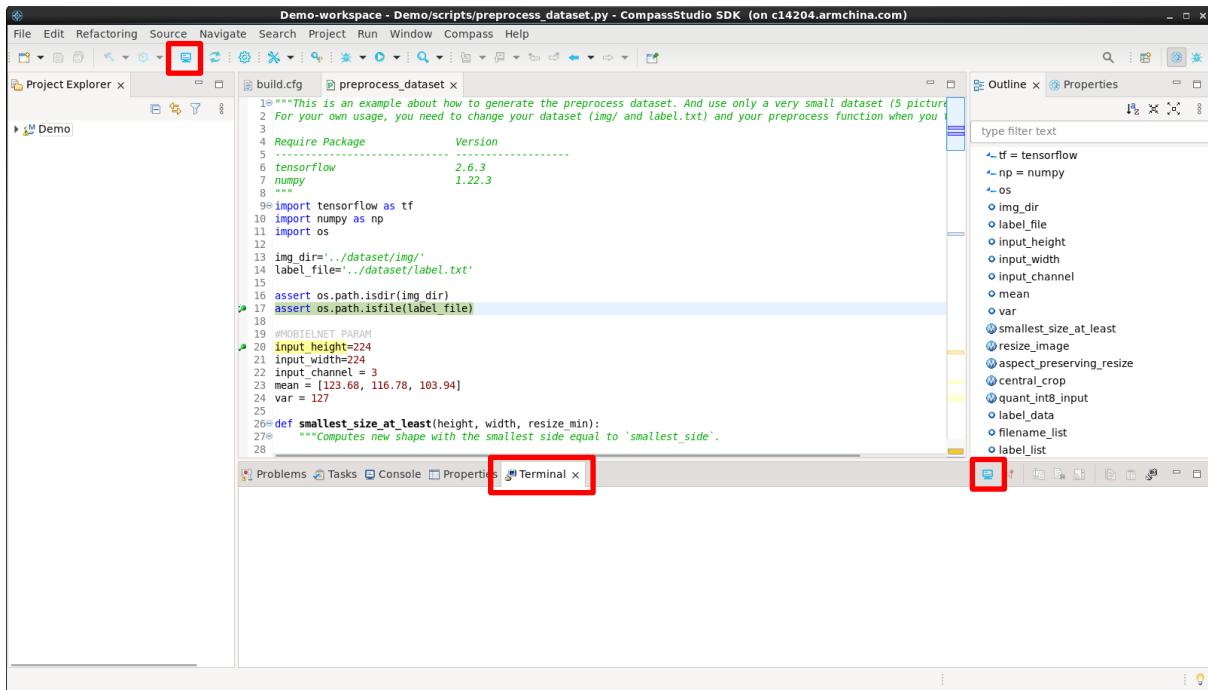
1. 打开 Terminal 视图，单击 **Window > Show View > Other**，如图 8-7 所示。

图 8-7: 打开 Terminal 视图

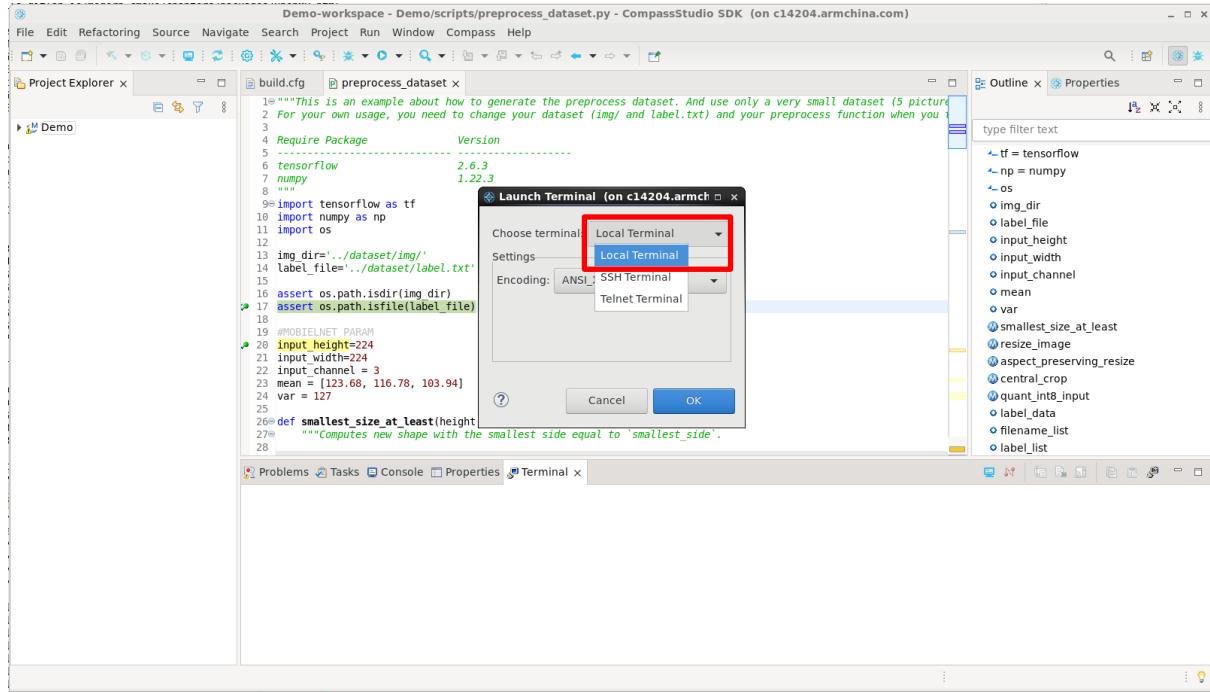


- 在打开的对话框中, 选择 Terminal > Terminal, 单击 Open 图标或者双击 Terminal, 即可打开 Terminal 视图, 如图 8-8 所示。

图 8-8: Terminal 视图

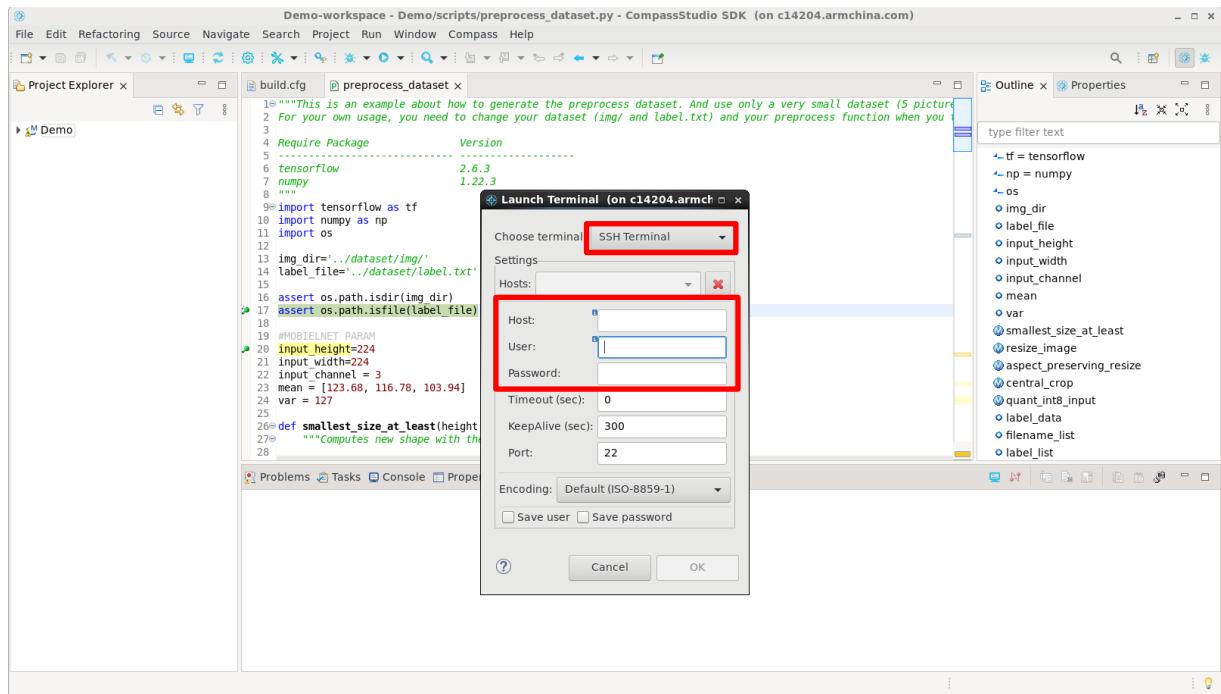


- 如图 8-8 所示, 单击 Open a Terminal 图标, 在弹出的对话框中选择 Terminal 类型创建一个或多个 Terminal 窗口, 如图 8-9 所示。

图 8-9: 创建本地 Terminal

- 要想在 CompassStudio 进行本地系统操作: 在 **Choose terminal** 下拉列表中, 选择 **Local Terminal**, 使用默认 Encoding 选项, 然后单击 **OK**, 即可打开本地的 Terminal 窗口。
- 要想在 CompassStudio 通过 SSH 远程连接 Server 进行操作: 在 **Choose terminal** 下拉列表中, 选择 **SSH Terminal**。在弹出的对话框中, 分别在 **Host** 文本框输入 SSH 远程 IP 地址, **User** 文本框输入用户名, **Password** 文本框输入密码, 其他选项使用默认设置即可。然后单击 **OK**, 即可创建 SSH 的 Terminal 窗口, 如图 8-10 所示。

图 8-10: 创建 SSH Terminal



9 常见问题

以下内容以 Debian/Ubuntu 发行版 (apt) 为例。如使用 Red Hat/Fedora 请在对应平台搜索软件包。

9.1 如何安装 Qt 和 Graphviz

Graphviz (OS Tool) :

```
sudo apt install graphviz
```

Qt: <https://download.qt.io/archive/qt/5.9/5.9.1/>

推荐使用以下脚本从官方网站下载，用图形向导安装 Qt 库：

```
sudo apt install libxcb-xinerama0
wget https://download.qt.io/archive/qt/5.9/5.9.1/qt-opensource-linux-x64-5.9.1.run
./qt-opensource-linux-x64-5.9.1.run
```

安装完后需要配置环境变量：

```
Export
LD_LIBRARY_PATH=/home/...../Qt5.9.1/5.9.1/gcc_64/lib:$LD_LIBRARY_PATH
```

9.2 如何设置 Compass SDK 运行环境

参考《Zhouyi Compass Software Technical Overview》文档。

9.3 CompssStudio 无法打开，报“缺少 SWT 相关库”错误

如果 Linux 环境缺少 SWT 相关的 lib，会报错：“Could not load SWT library”。您需要在 IT 协助下下载对应版本的 GTK 到系统环境中。如果您使用的是本地虚拟机环境，需要下载以下 lib，并拷贝到 “/home/UserName/.swt/lib/linux/x86_64” 即可。

- libswt-atk-gtk-xxx.so
- libswt-glx-gtk-xxx.so

- libswt-pi-gtk-xxx.so
- libswt.awt-gtk-xxx.so
- libswt-gnome-gtk-xxx.so
- libswt-webkit-gtk-xxx.so
- libswt-cairo-gtk-xxx.so
- libswt-gtk-xxx.so

9.4 如何在 CompassStudio 中配置 Anaconda 虚拟环境

假设在 Anaconda 创建了一个 python3.8 版本的虚拟环境（将其命名为 "Anaconda_Test"），将该虚拟环境加入到 CompassStudio 中的步骤为：

1. 在服务器上使用 “Anaconda_Test”的虚拟环境：

```
activate Anaconda_Test
```

2. 打开 CompassStudio，将该环境引入。依次单击 **Window > Preferences > PyDev > Interpreters > Python Interpreters** 即可进入设置 Python 解释器的界面。

如果想利用在 Anaconda 中建立的虚拟环境并在 CompassStudio 中使用，需要将新建虚拟环境中的解释器设置为 CompassStudio 现在使用的解析器，参考图 2-5 配置 Python 环境。此时，CompassStudio 解析路径为：Anaconda 安装路径 > envs > 环境名文件夹 (Anaconda_Test) > python.exe。

9.5 新建工程时，可能会报语法错误

新建工程时，CompassStudio 会自动打开算子 C 文件，需要索引 LLVM 的语法解析。根据主机性能，其所用时间不同。如果主机性能较差，可能会看到索引时产生错误，索引结束后错误会消失。如出现以上现象，请以编译时是否报错为准。

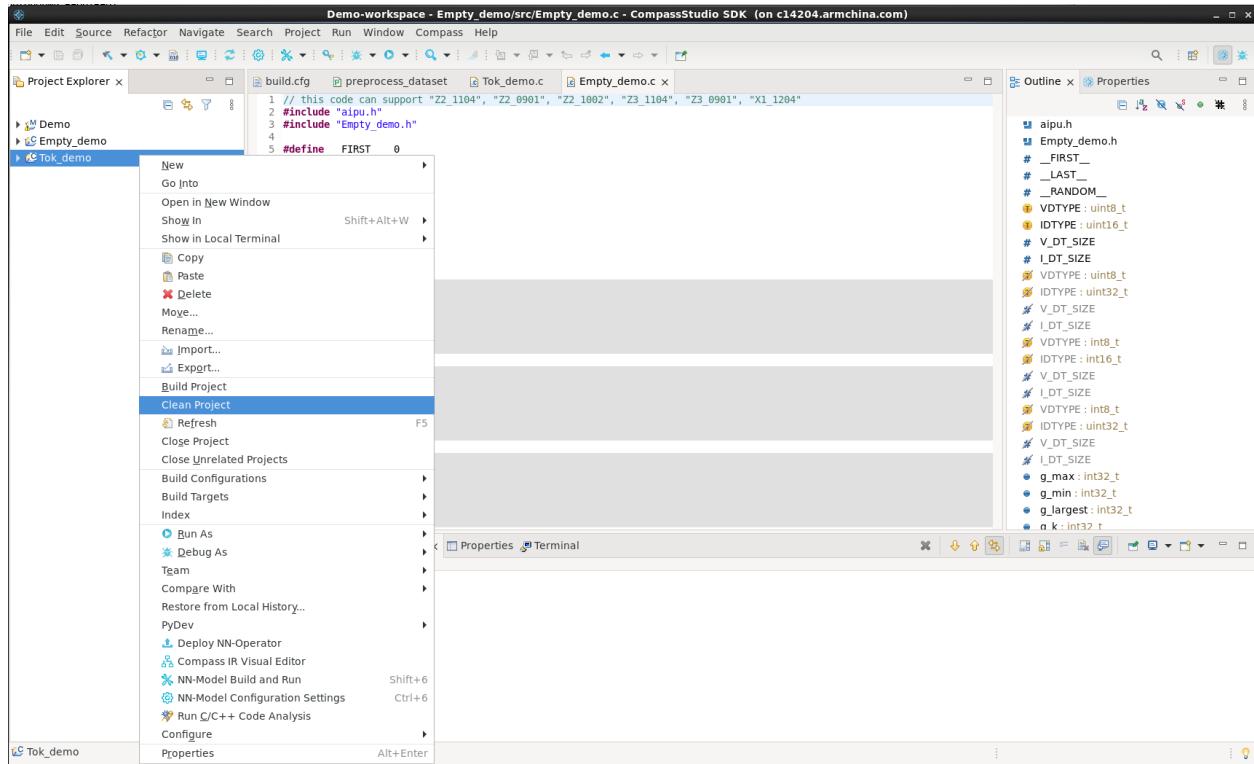
9.6 新增 Include 路径出现缓存

在 Include 页面新增路径时，可能会出现缓存的路径内容，此时单击 Workspace 或 File System 选中后可覆盖其内容。

9.7 算子工程编译出错

算子工程由于编译参数出现错误，或者您想加入新参数进行编译，需要先将工程清除。其操作如图 9-1 所示。

图 9-1：清除算子工程



9.8 找不到设置页面栏

有时可能因为打开的对话框太小，部分设置栏目被隐藏起来，此时可单击左右方向图标显示隐藏的栏目，如图 9-2 所示。

图 9-2: 显示隐藏的栏目