CANN算子库ATVOSS

极简、高效、高性能、高扩展的编程范式

作者：李东锋/沈敏

时间：2025-12-03

CANN

目录

背景

当前现状

- Tiling计算繁杂
- 指令灵活,操作维度多
- 性能调优困难
- 感知芯片差异

我们的思考

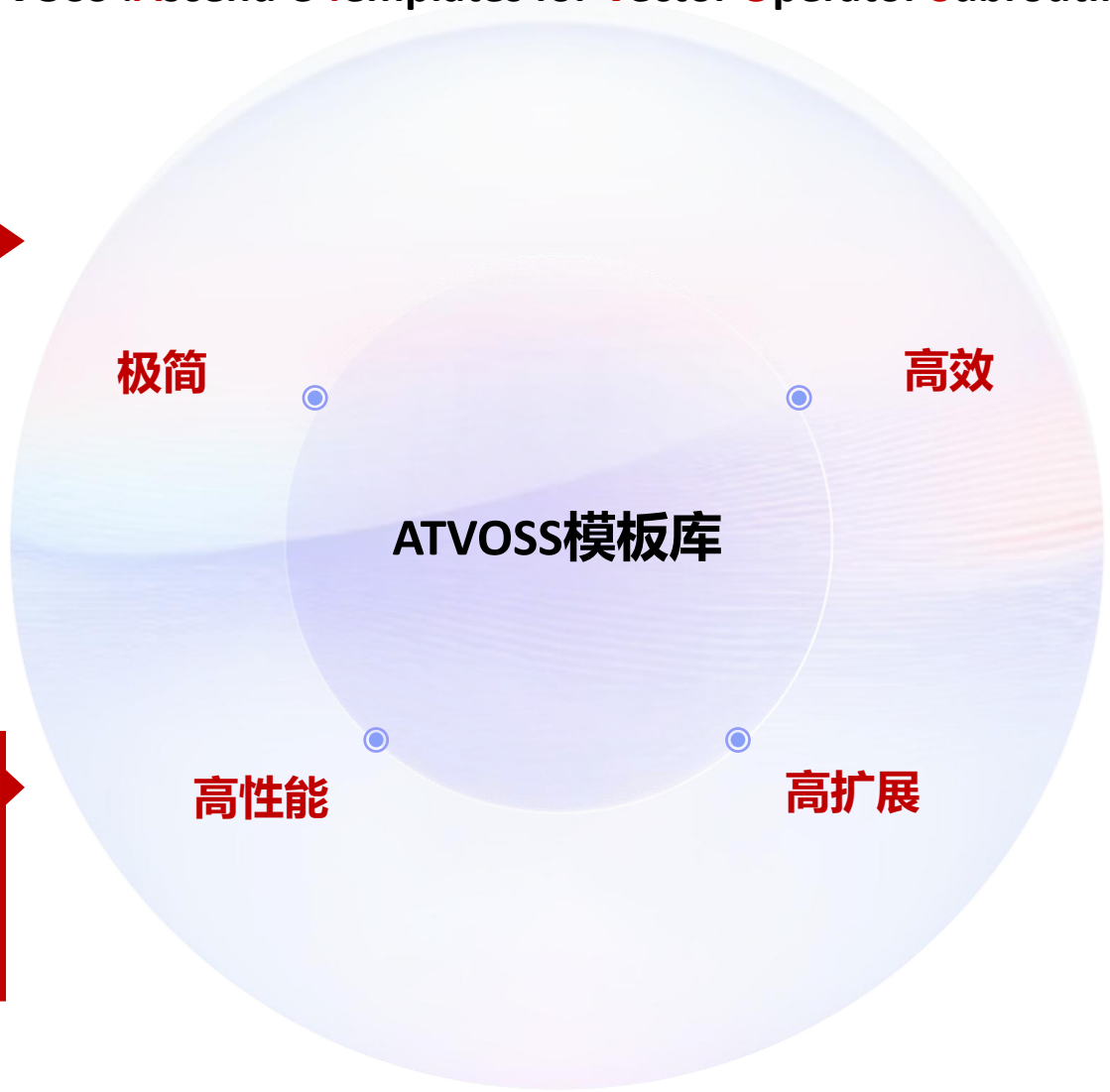
- 如何帮助用户高效写出一个高性能算子

我们要做什么样的东西

- 它是极简的
- 它是高效的
- 它性能要有保障
- 它应该支持扩展

设计目标：极简、高效、高性能、高扩展,全面提升算子开发易用性

ATVOSS :Ascend C Templates for Vector Operator Subroutines



采用极简表达式

编程界面简单易懂

采用分层架构，各层模板

灵活组装，完成算子功能

编译时完成所有常量计算

编译时内存分配

架构支持每分层独立使用

每层支持自定义

CANN

极简：采用极简表达式，编程界面简单易懂

Rms-norm计算公式：

$$\mathrm{RmsNorm}(x_i) = \frac{x_i}{\mathrm{Rms}(\mathbf{x})} g_i, \quad \text{where } \mathrm{Rms}(\mathbf{x}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

```
template<typename T>
struct RmsnormCompute {

    template <template <typename> class Tensor>
    __host_aicore__ constexpr auto Compute() const
    {
        auto x = PlaceHodler<1, Tensor<T>, Atvoss::Arguments::IN>();
        auto g = PlaceHodler<2, Tensor<T>, Atvoss::Arguments::IN>();
        auto y = PlaceHodler<3, Tensor<T>, Atvoss::Arguments::OUT>();
        auto tmp = PloceHolderTmpLike<1>(x);
        auto z   = PloceHolderTmpLike<2, Pattern::AR>(x);
        return (tmp = x * x,
                z = ReduceSum<Pattern::AR>(tmp),
                z = Divs(z, GetShape<Int<-1>>(x)),
                z = Sqrt(z),
                tmp = Broadcast<Pattern::AB>(z),
                tmp = x / tmp,
                y = g * tmp);
    }
};
```

抽象化计算表达界面：

延迟计算

- Device/Kenrel/Block层只记录表达式，形成类型化抽象语法树
- Tile层对表达式进行实际计算操作

CANN

极简：精简构建，快速验证

```
bisheng -x asc test.cpp -o test -I {atvoss_path}/include ……
```

- 提供一键编译运行的样例执行脚本

```
# 设置环境变量
source xxx/latest/bin/setenv.bash

# 切换到样例的目录
cd ../examples

# 编译执行命令
bash run_examples.sh cast
```

- 样例执行结果：

```
[root@localhost examples]# bash run_examples.sh cast
Executing with npu mode
Accuracy verification passed.
Sample cast passed!
```

- 采用**Host-device融合编译**技术
- 减少算子交付件
- 简化编译构建步骤

CANN

高效：灵活组装模板实现算子功能

⑩ **组装Block模板**：

BlockBuilder中嵌套自定义算子Compute

⑩ **组装Kernel模板**：

KernelBuilder中嵌套BlockOp

⑩ **组装Device模板**：

DeviceAdapter中嵌套KernelOp

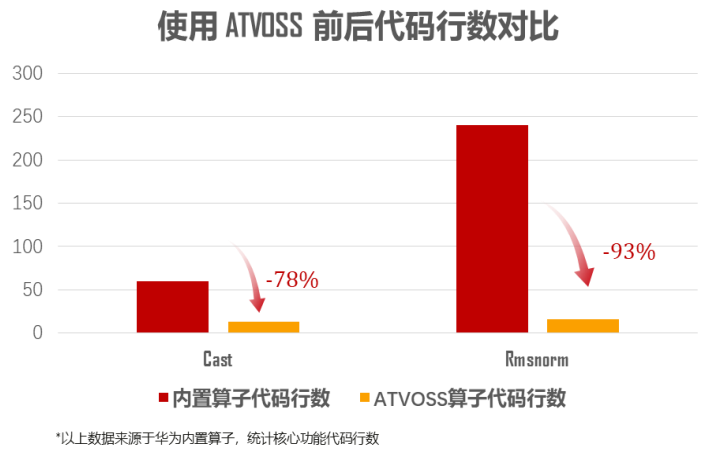
⑩ **自定义计算表达式**：与模板框架解耦，可独立验证

```
template<typename T>
struct RmsnormCompute {

    template <template <typename> class Tensor>
    __host_aicore__ constexpr auto Compute() const
    {
        ...
        return (tmp = x * x,
                z = ReduceSum<Pattern::AR>(tmp),
                z = Divs(z, GetShape<Int<-1>>(x)),
                z = Sqrt(z),
                tmp = Broadcast<Pattern::AB>(z),
                tmp = x / tmp,
                y = g * tmp);
    }
};
```

```
struct RmsnormCompute { ... };

// 组装Block层模板
using BlockOp = EleWise::BlockBuilder<
    Compute     = RmsnormCompute,
    Policy      = blockPolicy,
    ScheduleCfg = EleWise::BlockDefaultCfg,
    Schedule    = EleWise::BlockDefaultSchedule>;

// 组装Kernel层模板
using KernelOp = EleWise::KernelBuilder<
    BlockBuilder    = BlockOp,
    Policy          = kernelPolicy,
    ScheduleCfg     = EleWise::KernelDefaultCfg,
    Schedule        = EleWise::KernelDefaultSchedule>;

// 组装Device层模板
using DeviceOp = Atvoss::DeviceAdapter<KernelOp>;
```

高效：同一个计算表达式，使用不同模板实现多种算子场景

```
// 定义Compute
struct RmsnormCompute {
   template <template <typename> class Tensor>
    __host_aicore__ constexpr auto Compute() const
    {
        ....
        return (tmp = x * x,
                z = ReduceSum<Pattern::AR>(tmp),
                z = Divs(z, GetShape<Int<-1>>(x)),
                z = Sqrt(z),
                tmp = Broadcast<Pattern::AB>(z),
                tmp = x / tmp,
                y = g * tmp);
    }
};
```



https://gitcode.com/cann

⑩ 单Tile块切R场景, Reduce轴在Tile块内完成

```
// 组装Block层模板
using BlockOp = EleWise::BlockBuilder<
    Compute     = RmsnormCompute,
    Policy      = blockPolicy,
    ScheduleCfg   = EleWise::BlockDefaultCfg,
    Schedule    = EleWise::BlockDefaultSchedule>;

// 组装Kernel层模板
using KernelOp = EleWise::KernelBuilder<
    BlockBuilder    = BlockOp, ... >;
```

⑩ 单核切R场景, Reduce轴在Block内完成

```
// 组装Block层模板
using BlockOp = Reduce::BlockBuilder<
    Compute     = RmsnormCompute,
    Policy      = blockPolicy,
    ScheduleCfg   = Reduce::BlockDefaultCfg,
    Schedule    = Reduce::BlockDefaultSchedule>;

// 组装Kernel层模板
using KernelOp = EleWise::KernelBuilder<
    BlockBuilder    = BlockOp, ... >;
```

高性能：

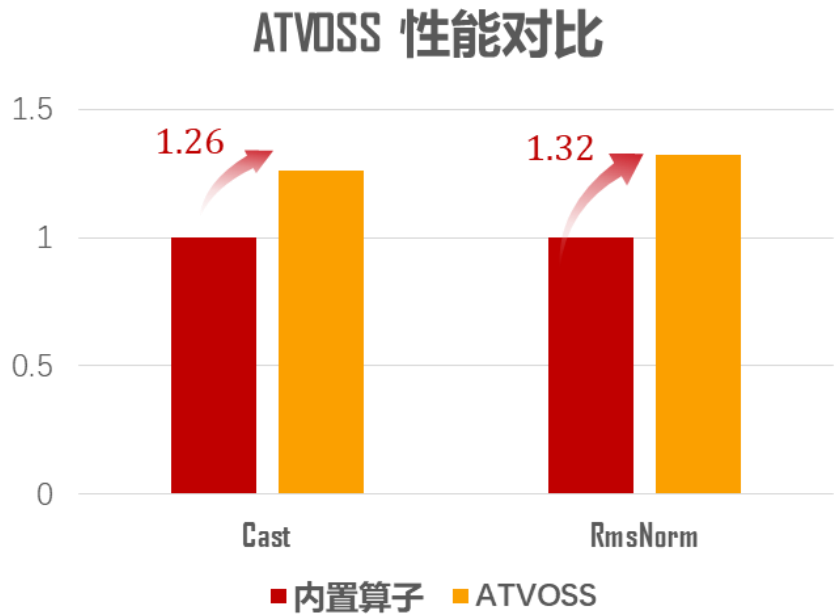
■ 模板自动推到最佳切分策略

1. 基于表达式计算存活节点，编译时获取最佳切分

2. 基于最佳切分决定切核策略

```
template<typename T>
struct RmsnormCompute {

    template <template <typename> class Tensor>
    __host_aicore__ constexpr auto Compute() const
    {
        ...
        return (tmp = x * x,
                z = ReduceSum<Pattern::AR>(tmp),
                z = Divs(z, GetShape<Int<-1>>(x)),
                z = Sqrt(z),
                tmp = Broadcast<Pattern::AB>(z),
                tmp = x / tmp,
                y = g * tmp);
    }
};
```

■ 定制场景，性能达内置算子的 1.2x以上



https://gitcode.com/cann

CANN

高扩展： 参数化配置、可移植

```
struct RmsnormCompute {
    __host_aicore__ constexpr auto Compute() const
    {
        ....
        return (tmp = x * x,
                z = ReduceSum<Pattern::AR>(tmp),
                z = Divs(z, GetShape<Int<-1>>(x)),
                z = Sqrt(z),
                tmp = Broadcast<Pattern::AB>(z),
                tmp = x / tmp,
                y = g * tmp);
    }
};
```

■ 表达式模板自由移植

通过Tile层与底层计算实现隔离，不同硬件平台可直接移植

■ 开放调优参数

支持Tile块大小配置

支持Kernel&block层Policy参数配置

```
// 配置Tile块大小
using TileShape = Atvoss::Shape<256, 1024>;

// 配置Block层静态Policy
static constexpr EleWise::BlockPolicy blockPolicy {
   190 * 1024,  // Maximum UB space size.
   TileShape{};
};

// 配置Kernel层静态Policy
static constexpr EleWise::KernelPolicy kernelPolicy {
   48, // 用户自定义最大启动核数
   EleWise::PolicySegment::UNIFORM_SEGMENT // 多核分配策略
 };
```


CANN

高扩展： 支持定制模板Schdule与Policy

Block层模板框架支持高度自定义

```
// 组装Block层模板
using BlockOp = EleWise::BlockBuilder<
    Compute     = AddMulCompute,
    Policy      = blockPolicy,
    ScheduleCfg = EleWise::BlockDefaultCfg,
    Schedule    = EleWise::BlockDefaultSchedule>;
```

- **Compute**：自定义计算表达式
- **Policy**：自定义切分策略
- **Schedule& ScheduleCfg** ：定制Block单核处理流程；

Kernel层模板框架支持高度自定义

```
// 组装Kernel层模板
using KernelOp = EleWise::KernelBuilder<
    BlockBuilder    = BlockOp,
    Policy          = kernelPolicy,
    ScheduleCfg     = EleWise::KernelDefaultCfg,
    Schedule        = EleWise::KernelDefaultSchedule>;
```

- **BlockBuilder** ：自定义计算表达式
- **Policy**：自定义切分策略
- **Schedule& ScheduleCfg** ：定制Kernel多核处理流程；

CANN

ATVOSS总体架构

CANN

ATVOSS编程界面

```
template<typename T>
struct AddMulCompute {

    template <template <typename> class Tensor>
    __host_aicore__ constexpr auto Compute() const
    {
        ...
        return (tmp = x * x,  ...);
    }
};

// 组装模板
using BlockOp = EleWise::BlockBuilder<AddMulCompute, blockPolicy, ... >;
using KernelOp = EleWise::KernelBuilder<BlockOp, kernelPolicy, ...>
using DeviceOp = Atvoss::DeviceAdapter<KernelOp>;

int main() {

// 构造输入数据，示例是固定输入数据值的， 实际可自定义生成数据
 Atvoss::Tensor<float> t1(dataV1/*data host ptr*/, {32, 32} /*shape*/);
 Atvoss::Tensor<float> t2(dataV2/*data host ptr*/, {32, 32} /*shape*/);
 Atvoss::Tensor<float> t3(dataV3/*data host ptr*/, {32, 32} /*shape*/);

// 生成入参信息
 Arguments arguments = ArgumentsBuilder{}.input({t1, t2}).output({t3}).build();

// 算子执行
 DeviceOp addMulOp();
 addMulOp.Run(arguments);
}
```

用户自定义计算表达式，描述计算逻辑

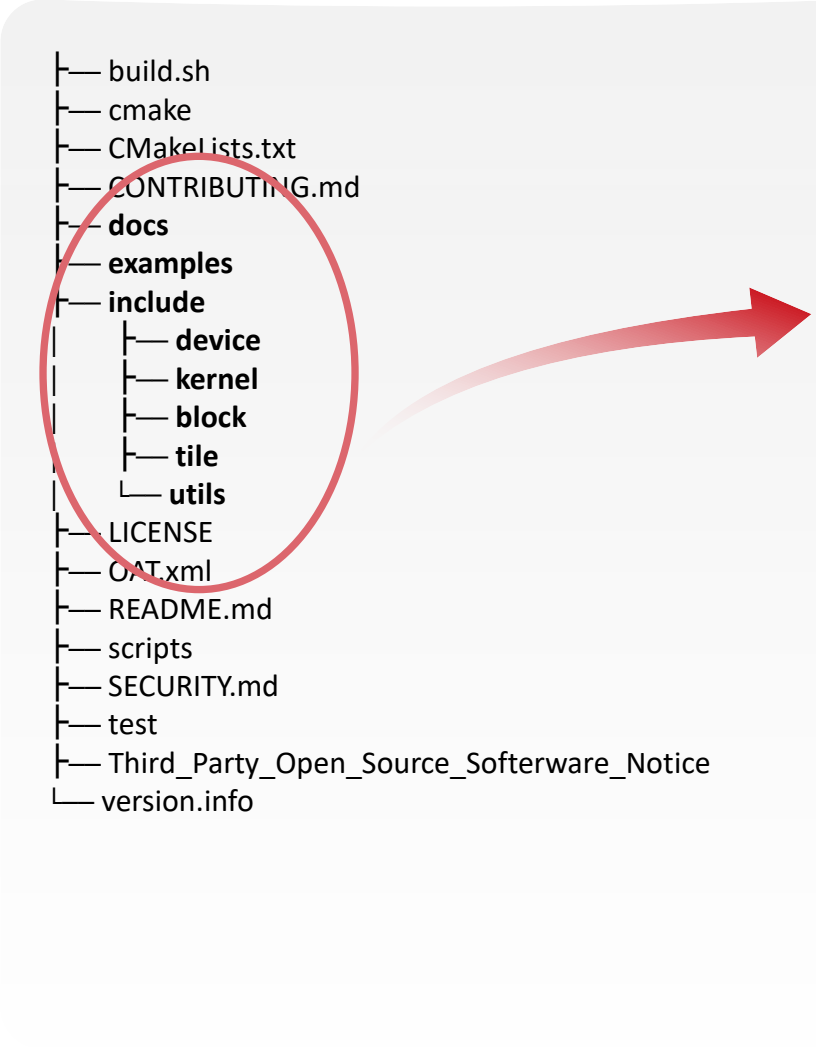
用户配置Tile块大小、UB最大内存等

用户自定义构造host侧输入输出数据

CANN

目录

ATVOSS仓介绍



- **Docs**：包含快速上手、用户使用指南等文档
- **Examples**：算子开发样例
- **include**：模板实现头文件库

  包含各层模板实现：device/kernel/block/tile

  包含计算表达模板等基础工具实现：utils

  …

代码仓链接：https://gitcode.com/cann/atvoss

ATVOSS未来规划：更丰富、更优、更易用

ATVOSS 正式开源

支持Elewise模板

2025

11/29 Nov.

2026

1月 Jan.

- 补充For-each类算子样例
- 发布Reduce单核切R模板及样例
- Compute API接口扩展、丰富

2026

3月 Mar.

- 发布Broadcast模板
- 发布Reduce高维模板及样例
- Reduce模板性能优化

2026

6月 Jun.

- 配套自动调优工具
- Broadcast模板性能优化

. . .

https://gitcode.com/cann

CANN

交流 & 贡献

ATVOSS 代码仓

https://gitcode.com/cann/atvoss

欢迎交流！欢迎贡献！欢迎共创合作！

Thank you.

社区愿景：打造开放易用、技术领先的AI算力新生态

社区使命：使能开发者基于CANN社区自主研究创新，构筑根深叶茂、跨产业协同共享共赢的CANN生态

Vision: Building an Open, Easy-to-Use, and Technology-leading AI Computing Ecosystem

Mission: Enable developers to independently research and innovate based on the CANN community and build a win-win CANN ecosystem with deep roots and cross-industry collaboration and sharing.

上CANN社区获取干货

关注CANN公众号获取资讯

https://gitcode.com/cann