曦云® 系列通用计算 GPU

mcEigen 用户指南

CSOG-23030-020-F3_V03　　2024-03-15

沐曦专有和三级保密信息　　本文档受 NDA 管控

声明

更新记录

版本	日期	更新说明
V03	2024-03-15	新增曦云®系列 GPU 产品信息
V02	2023-12-29	描述性修改及格式修正
V01	2023-10-16	正式版本首次发布

目录

1 介绍

Eigen 是一个用 C++ 实现的开源线性代数库。它速度快、适用性强，可以胜任从大规模的数值计算到简单的向量运算等任务。

1.1 安装

在安装 MXMACA® 工具包时，会将 Eigen 的头文件复制到系统上的标准 MXMACA 包含目录中。MXMACA 工具包的安装，参见《曦云系列通用计算 GPU 快速上手指南》。安装后，确保设置了环境变量 `MACA_PATH`。

```
export MACA_PATH=/opt/maca
```

这里的 `/opt/maca` 路径为 MXMACA 工具包的默认安装路径。如果安装时选择了其它路径，则将 `MACA_PATH` 设置为该路径。以下是与 mcEigen 相关的头文件所在目录

```
#header location
${MACA_PATH}/include/eigen3/Eigen
${MACA_PATH}/include/eigen3/unsupported/Eigen
```

Eigen 使用纯头文件库，也就是说所有源码都包含在头文件中。没有单独用于函数和类定义的 `.cpp` 文件——所有内容都在 `.h` 文件中。这就让使用 Eigen 变得非常简单。只需要简单的在自己代码的开头 `#include` 这些头文件就可以使用 Eigen。

1.2 Eigen 应用

要使用 Eigen，您只需要在 `Eigen` 子目录和 `unsupported/Eigen` 子目录中包含头文件。如果您想启用 GPU 功能，需要在包含 Eigen 头文件之前定义一个宏 EIGEN_USE_GPU。如果目标平台是 MXMACA，则在使用 mxcc 编译时还需要定义一个宏 EIGEN_USE_MACA。

以下是一个使用 Eigen 的简单示例：

```
// myEigenApp.cpp
#define EIGEN_USE_GPU
#define EIGEN_USE_MACA

#include <Eigen/Core>
#include <unsupported/Eigen/CXX11/Tensor>
#include <unsupported/Eigen/CXX11/src/Tensor/TensorGpuHipCudaDefines.h>

int main()
{
  constexpr int m_size = 128;
  constexpr int k_size = 128;
```

(下页继续)

(续上页)

```
  constexpr int n_size = 128;
  std::cout << "Testing for (" << m_size << "," << k_size << "," << n_size
↪<< ")" << std::endl;
  Eigen::Tensor<float, 3, Eigen::ColMajor> t_input(m_size, k_size, n_size);
  Eigen::Tensor<float, 3, Eigen::ColMajor> t_result(m_size, k_size, n_
↪size);
  Eigen::Tensor<float, 3, Eigen::ColMajor> t_result_gpu(m_size, k_size, n_
↪size);

  t_input.setRandom();

  std::size_t t_input_bytes = t_input.size()  * sizeof(float);
  std::size_t t_result_bytes = t_result.size() * sizeof(float);

  float* d_t_input;
  float* d_t_result;

  gpuMalloc((void**)(&d_t_input), t_input_bytes);
  gpuMalloc((void**)(&d_t_result), t_result_bytes);

  gpuMemcpy(d_t_input, t_input.data(), t_input_bytes,
↪gpuMemcpyHostToDevice);

  Eigen::GpuStreamDevice stream;
  Eigen::GpuDevice gpu_device(&stream);

  Eigen::TensorMap<Eigen::Tensor<float, 3, Eigen::ColMajor> >
      gpu_t_input(d_t_input, Eigen::array<int, 3>(m_size, k_size, n_size));
  Eigen::TensorMap<Eigen::Tensor<float, 3, Eigen::ColMajor> >
      gpu_t_result(d_t_result, Eigen::array<int, 3>(m_size, k_size, n_
↪size));

  gpu_t_result.device(gpu_device) = gpu_t_input.cumsum(1);
  t_result = t_input.cumsum(1);

  bool ret = true;
  gpuMemcpy(t_result_gpu.data(), d_t_result, t_result_bytes,
↪gpuMemcpyDeviceToHost);
  for (Eigen::DenseIndex i = 0; i < t_result.size(); i++) {
    if (fabs(t_result(i) - t_result_gpu(i)) < 1e-4f) {
      continue;
    }
    if (Eigen::internal::isApprox(t_result(i), t_result_gpu(i), 1e-4f)) {
      continue;
    }
    ret = false;
    std::cout << "mismatch detected at index " << i << ": " << t_result(i)
              << " vs " <<  t_result_gpu(i) << std::endl;
  }
  if (ret) {
    std::cout << "the result is correct" << std::endl;
  } else {
    std::cout << "the result is mismatched" << std::endl;
  }
  gpuFree((void*)d_t_input);
  gpuFree((void*)d_t_result);
```

(下页继续)

(续上页)

```
  return 0;
}
```

假设上面的示例包含在名为 myEigenApp.cpp 的文件中，您可以在 Linux 上使用以下命令进行编译：

```
export MACA_PATH=/opt/maca
export MACA_CLANG_PATH=${MACA_PATH}/mxgpu_llvm/bin
export LD_LIBRARY_PATH=${MACA_PATH}/lib:${LD_LIBRARY_PATH}

${MACA_CLANG_PATH}/mxcc -x maca myEigenApp.cpp -o myEigenApp -I ${MACA_
↪PATH}/include/eigen3
```

这里的 /opt/maca 路径为 MXMACA 工具包的默认安装路径。如果安装时选择了其它路径，则将 MACA_PATH 设置为该路径。

1.3 工程项目中使用 mcEigen

以下是一个 CMakeLists.txt 的简单示例，用于编译上面的 myEigenApp.cpp。

```
# CMakeLists.txt
cmake_minimum_required(VERSION 3.16)
project(example)

# maca toolkits path
set(MACA_PATH $ENV{MACA_PATH})
set(CMAKE_C_COMPILER ${MACA_PATH}/mxgpu_llvm/bin/mxcc)
set(CMAKE_CXX_COMPILER ${MACA_PATH}/mxgpu_llvm/bin/mxcc)

# eigen path in maca toolkits
set(eigen_src_dir ${MACA_PATH}/include/eigen3)

set(example_target myEigenApp)
set(example_src myEigenApp.cpp)

add_compile_options( -x maca -fPIC
                     -DEIGEN_USE_GPU
                     -DEIGEN_USE_MACA
                     --maca-device-lib-path=${MACA_PATH}/lib
                     --maca-device-lib=maca_mathlib.bc
                     --maca-device-lib=maca_kernellib.bc
                     --maca-host-lib-path=${MACA_PATH}/lib
                     --maca-host-lib=maca_mathlib_host.bc
)

add_executable(${example_target} "${example_src}")
target_include_directories(${example_target} PRIVATE ${eigen_src_dir})
```

把 myEigenApp.cpp 和 CMakeLists.txt 放在同一目录下，然后执行命令：

```
export MACA_PATH=/opt/maca

mkdir build
cmake -B build .
cmake --build build -j
```

这里的 /opt/maca 路径为 MXMACA 工具包的默认安装路径。如果安装时选择了其它路径，则将 MACA_PATH 设置为该路径。编译结束后，二进制文件 myEigenApp 就会出现在 build 文件夹中。

2 使用 Eigen API

2.1 模块以及头文件

Eigen 库可以分为一个 Core 模块和几个其他模块。每个模块都有一个相对应的头文件，只有包含对应的头文件才能使用对应的模块。为方便同时访问多个模块，我们提供了 Dense 和 Eigen 头文件。Eigen 头文件的内容如下：

- Core: 包含矩阵和数组类, 基本线性代数 (包括三角函数和自伴乘积相关), 以及相关数组操作
- Geometry: 包括变换, 平移、缩放、二维旋转、三维旋转 (四元数, 角轴相关)
- LU: 有关求逆、求行列式、LU 分解解算器 (FullPivLU, PartialPivLU)
- Cholesky: 包含 LLT 和 LDLT 的 Cholesky 因式分解法
- Householder: Householder 变换; 这个模块供几何线性代数模块使用
- SVD: SVD 奇异值分解, 最小二乘解算器解决奇异值分解 (JacobiSVD, BDCSVD)
- QR: QR 分解解算器 (HouseholderQR, ColPivHouseholderQR, FullPivHouseholderQR)
- Eigenvalues: 特征值和特征向量分解 (EigenSolver, SelfAdjpintEigenSolver, ComplexEigenSolver)
- Sparse: 稀疏矩阵存储以及相关线性代数 (SparseMatrix, SparseVector)
- Dense: 包含 Core, Geometry, LU, Cholesky, SVD, QR 和 Eigenvalues 模块的头文件
- Eigen: 包含 Dense 和 Sparse 的头文件 (上述所有模块)