未对齐数组的断言说明

密集矩阵和数组操作»对齐问题

你好！您看到此网页是因为您的程序因断言失败而终止，如下所示：

```
my_program: path/to/eigen/Eigen/src/Core/DenseStorage.h:44:
Eigen::internal::matrix_array<T, Size, MatrixOptions,
Align>::internal::matrix_array()
[with T = double, int Size = 2, int MatrixOptions = 2, bool Align = true]:
Assertion `(reinterpret_cast<size_t>(array) & (sizemask)) == 0 && "this
assertion
is explained here: http://eigen.tuxfamily.org/dox-
devel/group__TopicUnalignedArrayAssert.html
     READ THIS WEB PAGE !!! ****"' failed.
```

此问题有 4 个已知原因。如果您只能使用最近的编译器（例如 GCC>=7、clang>=5、MSVC>=19.12）来定位**[c++17]**，那么您很幸运：启用 c++17 应该就足够了（如果没有，请向我们报告）。否则，请继续阅读以了解这些问题并了解如何解决这些问题。

在我自己的代码中，问题的原因在哪里？

首先，您需要找出在您自己的代码中该断言是从哪里触发的。乍一看，错误消息似乎没有帮助，因为它指的是 Eigen 中的一个文件！但是，由于您的程序崩溃了，如果您可以重现崩溃，则可以使用任何调试器获得回溯。例如，如果您使用 GCC，则可以按如下方式使用 GDB 调试器：

```
$ gdb ./my_program          # Start GDB on your program
> run                       # Start running your program
...                         # Now reproduce the crash!
> bt                        # Obtain the backtrace
```

现在您已经准确地知道问题发生在您自己的代码中的哪个位置，请继续阅读以了解您需要更改的内容。

原因 1：具有特征对象作为成员的结构

如果你有这样的代码，

```
class Foo
{
  //...
  Eigen::Vector4d v;
  //...
};
//...
Foo *foo = new Foo;
```

那么你需要阅读这个单独的页面：具有特征成员的结构。

请注意，此处， Eigen::Vector4d 仅用作示例，更一般地，所有固定大小的可矢量化特征类型都会出现此问题。

原因 2：STL Containers 或手动内存分配

如果您将 STL 容器（例如 std::vector、std::map、...）与 Eigen 对象或包含 Eigen 对象的类一起使用，如下所示，

```
std::vector<Eigen::Matrix2d> my_vector;
struct my_class { ... Eigen::Matrix2d m; ... };
std::map<int, my_class> my_map;
```

那么您需要阅读这个单独的页面：将 STL 容器与 Eigen 一起使用。

请注意，这里， Eigen::Matrix2d 仅用作示例，更一般地，所有固定大小的可矢量化特征类型和结构都会出现问题，这些特征对象为 member。

任何类/函数绕过运算符 new 来分配内存，即通过执行自定义内存分配，然后调用放置 new 运算符，都会出现相同的问题。例如，这就是典型的情况 `std::make_shared` 或 `std::allocate_shared` 解决方案是使用对齐的分配器，如STL 容器的解决方案中详述的那样。

原因 3：按值传递 Eigen 对象

如果您的代码中的某些函数正在获取按值传递的特征对象，如下所示，

```
void func(Eigen::Vector4d v);
```

那么你需要阅读这个单独的页面：Passing Eigen objects by value to functions。

请注意，此处， Eigen::Vector4d 仅用作示例，更一般地，所有固定大小的可矢量化特征类型都会出现此问题。

原因 4：编译器对堆栈对齐做出错误假设（例如 Windows 上的 GCC）

对于在 Windows 上使用 GCC（如 MinGW 或 TDM-GCC）的人来说，这是必读的。如果您在声明局部变量的无害函数中遇到此断言失败，如下所示：

```
void foo()
{
  Eigen::Quaternionf q;
  //...
}
```

那么你需要阅读这个单独的页面：编译器对堆栈对齐做出错误的假设。

请注意，这里，Eigen::Quaternionf仅用作示例，更一般地，所有固定大小的可矢量化 Eigen 类型都会出现此问题。

此断言的一般解释

固定大小的可矢量化特征对象必须绝对在正确对齐的位置创建，否则处理它们的 SIMD 指令将崩溃。例如，SSE/NEON/MSA/Altivec/VSX 目标将需要 16 字节对齐，而 AVX 和 AVX512 目标可能分别需要最多 32 和 64 字节对齐。

Eigen 通常会为您处理这些对齐问题，方法是为它们设置对齐属性并重载它们的 `operator new`.

然而，有一些极端情况会覆盖这些对齐设置：它们是此断言的可能原因。

我不关心最佳矢量化，我该如何摆脱那些东西？

三种可能：

- 使用Matrix、Array、Quaternion等对象的`DontAlign`选项给你带来麻烦。这样 Eigen 不会尝试过度对齐它们，因此不会假设任何特殊对齐。不利的一面是，您将为它们支付未对齐加载/存储的成本，但在现代 CPU 上，开销是null 或marginal。请参见此处的示例。
- 将EIGEN_MAX_STATIC_ALIGN_BYTES定义为 0。这将禁用所有 16 字节（及以上）静态对齐代码，同时保持 16 字节（或以上）堆对齐。这具有通过未对齐存储（由EIGEN_UNALIGNED_VECTORIZE控制）矢量化固定大小对象（如 Matrix4d）的效果，同时保持动态大小对象（如 MatrixXd）的矢量化不变。在 64 字节系统上，您还可以将其定义为 16 以仅禁用 32 和 64 字节的过度对齐。但请注意，这会破坏 ABI 与静态对齐的默认行为的兼容性。
- 或者同时定义EIGEN_DONT_VECTORIZE和`EIGEN_DISABLE_UNALIGNED_ARRAY_ASSERT`. 这将保留 16 字节（或以上）对齐代码，从而保留 ABI 兼容性，但完全禁用矢量化。

如果您想知道为什么定义`EIGEN_DONT_VECTORIZE`本身不会禁用 16 字节（或以上）对齐和断言，这里是解释：

它不会禁用断言，否则在启用矢量化时，无需矢量化即可正常运行的代码会突然崩溃。它不会禁用 16 字节（或以上）对齐，因为这意味着矢量化和非矢量化代码不是相互 ABI 兼容的。这种 ABI 兼容性非常重要，即使对于仅开发内部应用程序的人来说也是如此，例如，人们可能希望在同一个应用程序中具有矢量化路径和非矢量化路径。

如何检查我的代码在对齐问题方面是否安全？

不幸的是，在 C++ 中没有可能在编译时检测到上述任何缺点（尽管静态分析器变得越来越强大并且可以检测到其中的一些）。即使在运行时，我们所能做的就是捕获无效的未对齐分配并触发本页开头提到的显式断言。因此，如果您的程序在具有某些给定编译标志的给定系统上运行良好，那么这并不能保证您的代码是安全的。例如，在大多数 64 位系统上，缓冲区在 16 字节边界上对齐，因此，如果您不启用 AVX 指令集，那么您的代码将运行良好。另一方面，如果移动到更奇特的平台，或者启用默认情况下需要 32 字节对齐的 AVX 指令，则相同的代码可能会断言。

不过，情况并非没有希望。假设您的代码被单元测试很好地覆盖，那么您可以通过将其链接到仅返回 8 字节对齐缓冲区的自定义 malloc 库来检查其对齐安全性。这样，所有对齐缺陷都应该弹出。为此，您还必须使用EIGEN_MALLOC_ALREADY_ALIGNED=0编译您的程序。