将 STL 容器与 Eigen 结合使用

执行摘要

如果您仅使用足够新的编译器（例如，GCC>=7、clang>=5、MSVC>=19.12）在**[c++17]**模式下进行编译，那么编译器会处理一切，您可以停止阅读。

否则，在固定大小的可矢量化特征类型或具有此类类型成员的类上使用 STL 容器需要使用过度对齐的分配器。也就是说，分配器能够分配 16、32 甚至 64 字节对齐的缓冲区。Eigen 确实提供了一个随时可用的：aligned_allocator。

在**[c++11] 之前**，如果要使用`std::vector`容器，则还必须`#include <Eigen/StdVector>`.

这些问题仅在固定大小的可矢量化特征类型和结构中出现，并且结构具有像 member 这样的特征对象。对于其他 Eigen 类型，例如 Vector3f 或 MatrixXd，在使用 STL 容器时无需特别注意。

使用对齐的分配器

STL 容器采用一个可选的模板参数，即分配器类型。在固定大小的可矢量化特征类型上使用 STL 容器时，您需要告诉容器使用一个分配器，该分配器将始终在 16 字节对齐（或更多）位置分配内存。幸运的是，Eigen 确实提供了这样一个分配器：Eigen::aligned_allocator。

例如，代替

```
std::map<int, Eigen::Vector4d>
```

你需要使用

```
std::map<int, Eigen::Vector4d, std::less<int>,
         Eigen::aligned_allocator<std::pair<const int, Eigen::Vector4d> > >
```

请注意，第三个参数`std::less<int>`只是默认值，但我们必须包含它，因为我们要指定第四个参数，即分配器类型。

std::vector 的情况

本节仅适用于 c++98/03 用户。**[c++11]**（或以上）用户可以停止阅读这里。

所以在c++98/03中，`std::vector`由于标准中的一个bug（解释如下），情况会更加复杂。为了解决这个问题，我们必须将它专门用于Eigen::aligned_allocator类型。在实践中，您**必须**使用Eigen::aligned_allocator（不是另一个对齐的分配器）**和**#include <Eigen/StdVector>。

下面是一个例子：

```
#include<Eigen/StdVector>
/* ... */
std::vector<Eigen::Vector4f,Eigen::aligned_allocator<Eigen::Vector4f> >
```

说明： `resize()` 方法 `std::vector` 接受一个 `value_type` 参数（默认为 `value_type()`）。因此，使用 `std::vector<Eigen::Vector4d>`，某些 Eigen::Vector4d 对象将按值传递，这会丢弃任何对齐修饰符，因此可以在未对齐的位置创建 Eigen::Vector4d。为了避免这种情况，我们看到的唯一解决方案是专门 `std::vector` 对其进行轻微修改，这里的 Eigen::Vector4d 能够正确处理这种情况。

另一种选择 - 专门用于特征类型的 std::vector

作为上述推荐方法的替代方法，您可以选择为需要对齐的特征类型专门化 std::vector 。优点是你不需要用Eigen::aligned_allocator来声明 std::vector 。另一方面的一个缺点是需要在所有 `std::vector<Vector2d>` 使用eg 的代码段之前定义特化。否则，在不知道专业化的情况下，编译器将使用默认值编译该特定实例，`std::allocator` 并且您的程序很可能会崩溃。

下面是一个例子：

```
#include<Eigen/StdVector>
/* ... */
EIGEN_DEFINE_STL_VECTOR_SPECIALIZATION(Matrix2d)
std::vector<Eigen::Vector2d>
```