Eigen 和多线程

使 Eigen 并行运行

某些 Eigen 算法可以利用硬件中存在的多个内核。为此，在编译器上启用 OpenMP 就足够了，例如：

- GCC：`-fopenmp`
- ICC：`-openmp`
- MSVC：检查构建属性中的相应选项。

您可以使用 OpenMP API 或 Eigen 的 API 使用以下优先级来控制将使用的线程数：

```
OMP_NUM_THREADS=n ./my_program
omp_set_num_threads(n);
Eigen::setNbThreads(n);
```

除非`setNbThreads`被调用，否则 Eigen 使用 OpenMP 指定的线程数。您可以通过调用来恢复此行为`setNbThreads(0);`。您可以查询将用于的线程数：

```
n = Eigen::nbThreads();
```

您可以通过定义EIGEN_DONT_PARALLELIZE预处理器令牌在编译时禁用 Eigen 的多线程。

目前，以下算法可以利用多线程：

- 一般稠密矩阵 - 矩阵乘积
- PartialPivLU
- row-major-sparse * dense vector/matrix products
- 共轭梯度与`Lower|Upper`作为`UpLo`模板参数。
- 具有行主稀疏矩阵格式的BiCGSTAB。
- 最小二乘共轭梯度

> 警告
>
> 在大多数操作系统上，将线程数量限制为物理内核数量**非常重要**，否则预计会显着减速，尤其是对于涉及密集矩阵的操作。

的确，超线程的原理是在单核上以交错的方式运行多个线程（大多数情况下是2个）。但是，Eigen 的矩阵-矩阵产品内核经过全面优化，已经利用了近 100% 的 CPU 容量。因此，没有空间在单个内核上运行多个此类线程，并且由于缓存污染和其他开销来源，性能会显着下降。在阅读的这个阶段，您可能想知道为什么 Eigen 不限制物理内核的数量？这仅仅是因为 OpenMP 不允许知道物理内核的数量，因此 Eigen 将启动与OpenMP 报告的*内核*一样多的线程。

在多线程应用程序中使用 Eigen

如果您自己的应用程序是多线程的，并且多个线程调用 Eigen，那么您必须**在**创建线程**之前**通过调用以下例程来初始化 Eigen：

```cpp
#include <Eigen/Core>

int main(int argc, char** argv)
{
    Eigen::initParallel();

    ...
}
```

> 注意
>
> 使用 Eigen 3.3 和完全符合 C++11 的编译器（即线程安全静态局部变量初始化），则调用 `initParallel()` 是可选的。

> 警告
>
> 请注意，所有生成随机矩阵的函数**都不是可**重入的，也不是线程安全的。这些包括 DenseBase::Random()和DenseBase::setRandom()尽管调用了 `Eigen::initParallel()`. 这是因为这些函数是基于 `std::rand` 不可重入的。对于线程安全的随机生成器，我们推荐使用 c++11 随机生成器（示例）或 `boost::random`.

如果您的应用程序与 OpenMP 并行化，您可能希望禁用 Eigen 自己的并行化，如上一节所述。

> 警告
>
> 将 OpenMP 与可能引发异常的自定义标量类型一起使用可能会在引发异常时导致意外行为。