# 充分利用xgboost和LightGBM的速度：编译器、CPU固定

目前，xgboost和LightGBM是两种性能最好的大型数据集机器学习算法（在速度和度量性能方面）。它们可以很好地扩展到十亿个观测和/或元素（例如：声誉数据集）。

xgboost和LightGBM主要是为了提高速度：与其等你的神经网络在数小时后完成工作，不如以高精度快速迭代，尝试更多不同的东西。

然而，尽管它们可以用于大型数据集，可伸缩性的问题是：xgboost和LightGBM的伸缩性如何？他们喜欢高频磁芯还是更多磁芯？

* xgboost exact喜欢多核和高频，两者都有偏好
* xgboost快速直方图需要高频
* LightGBM喜欢多核和高频，更喜欢高频

正如我们已经知道这个问题的答案，我们将寻找一个更奇特的情况：改变编译器和固定CPU。

通过将编译器从MinGW交换到Visual Studio，xgboost和LightGBM是否更快？CPU固定是件好事吗？

这一点在中也得到了部分回答。因此，我们回到Windows机器上做一些基准测试。

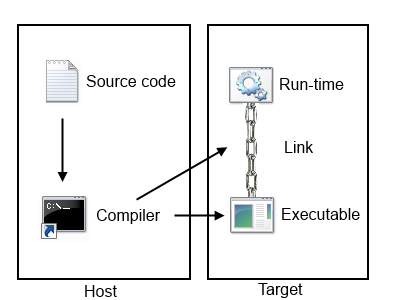
交互式文档：

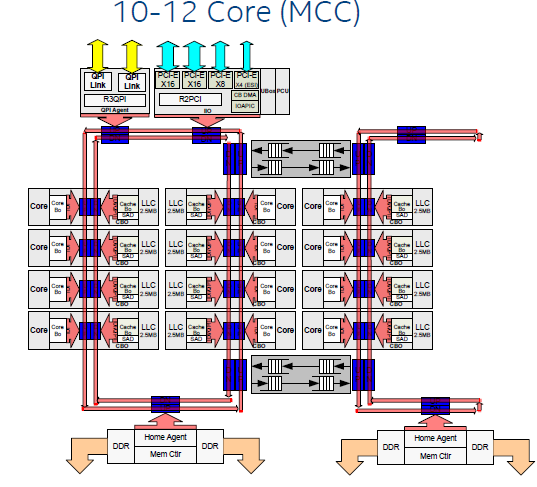
* [xgboost and LightGBM raw data](https://benchmark.laurae.design/speed_r_perf_analysis.html)
* [Visual Studio 2017 vs MinGW 4.9](https://benchmark.laurae.design/speed_r_vs_mingw.html)
* [CPU Roaming vs CPU Pinning](https://benchmark.laurae.design/speed_r_roaming_pinning_cpu.html)
* [GPU xgboost raw data](https://benchmark.laurae.design/speed_r_perf_gpu_analysis.html)

最后，对GPU-xgboost进行了开放。

### 编译器定义和CPU固定的快速回顾

#### 定义编译器和CPU固定





* 编译器：编译器将源语言的代码转换为目标语言的代码（通常用于生成可执行文件）。它们类似于翻译人员，我们都知道翻译人员的表现水平并不相同：有些人提供的是乱七八糟的单词，有些人提供的是出色的翻译，这反过来又会使你对单词的解释变慢或变快。
* CPU固定：CPU固定是将进程（或线程）绑定到特定范围的CPU核心。这样，如果没有CPU固定，进程就不会像它那样容易地漫游到任何地方。当进程在CPU之间漫游时，它会导致明显更高的RAM和缓存延迟：对于多插槽CPU，这甚至更为严重。

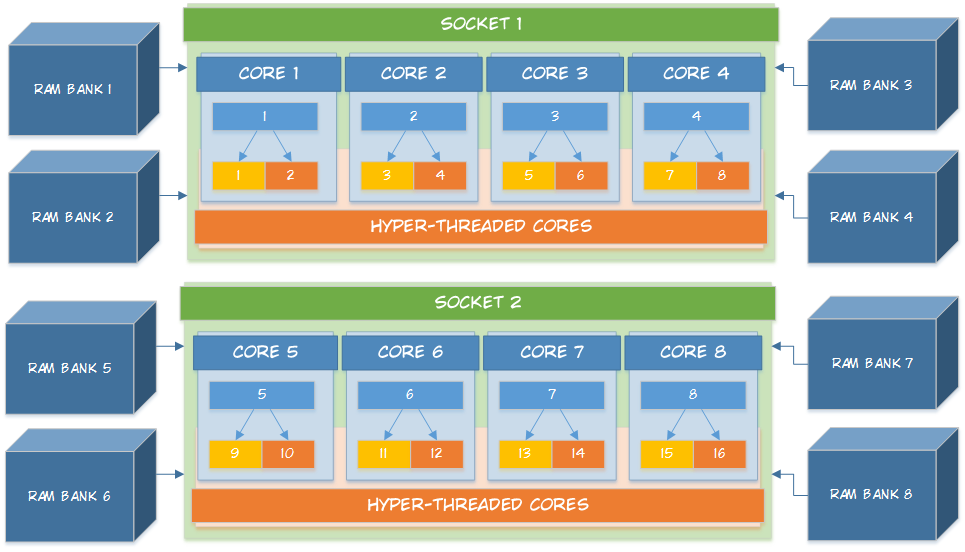
CPU pinning也被称为CPU affinity，尽管措辞不准确（“affinity”可能是指“preference”，尽管在本例中并非如此：它是“此进程使用此范围，并且仅使用此范围的CPU核心”）。

#### 对差异进行基准测试

我们将对编译器和CPU固定之间的差异进行基准测试，对于服务器上的每个可用线程数（1到56）：

* 要测试的两个编译器：Visual Studio（Windows的本机）和MinGW（gcc）
* 两种CPU行为：CPU漫游（无固定）和CPU固定（通过套接字，然后通过物理核心，然后通过超线程核心）。

后者的意思是：如果我们有2个套接字，每个套接字上有4个物理内核，并且已被超线程激活，我们将尝试在一个套接字中包含所有CPU，首先添加物理（黄色）内核，然后添加逻辑（橙色）内核：



我们正在以下环境下对xgboost和LightGBM进行基准测试：

* CPU：双Intel Xeon E5–2697v3（14核，28线程，3.6 GHz单线程，3.1 GHz多线程）
* 内存：128GB内存DDR4 2133 MHz
* GPU:无
* 操作系统：Windows Server 2012 R2数据中心，无熔毁/幽灵补丁
* R版本：默认3.4.3
* 编译器：Visual Studio 2017，MinGW 4.9（R）
* xgboost:commit 3f3f54b（2018年1月16日下午5:16 GMT+1）
* LightGBM:commit 3dc5716（2018年1月18日，格林尼治标准时间上午2:16+1）

数据集：

* [Kaggle Bosch training dataset](https://www.kaggle.com/c/bosch-production-line-performance)
* 观察次数：1183747
* 功能数：969
* 稀疏度：约81%

算法参数：

* boosting迭代次数：200
* 学习率：0.05
* 最大深度：8
* 最大叶数：255
* 最大箱数：255
* 最小黑森：1
* 仅限xgboost：快速直方图，按深度
* 仅限LightGBM：最小分割损耗为1（由于损耗导向优化）

每次跑步至少重复两次，最多10次。运行了大约1个星期来运行基准测试，多亏了这么多线程！!!

### 基准结果

#### 提醒：xgboost和LightGBM根本没有线性缩放。

xgboost比单个线程快154%，而LightGBM比单个线程快1116%。

如果你有工作站

* 如果有56个线程，不要期望56个线程的效率比1个线程高5500%（训练速度不会快55倍）。
* 如果你有28个内核，不要期望28个线程比1个线程效率高2700%（它训练速度不会快27倍）。
* 如果您有一个小的数据集，不要期望很多线程能够很好地伸缩（它会负伸缩）。

显示以下采用最佳情况方案（Visual Studio，漫游CPU）的结果：

#### 编译器性能

到目前为止，Visual Studio是Windows上的编译器。值得安装以获得尽可能快的训练速度。

使用漫游CPU：

使用CPU固定：

#### CPU固定性能

CPU钉扎显著提高了xgboost与MinGW的性能。否则，我们会看到性能下降。

故事士气：

* 如果在MinGW中使用xgboost，请使用CPU pinning。
* 另一种情况是：如果您在同一台计算机上训练并行xgboost和LightGBM，请锁定CPU，以确保CPU缓存效果能够正确触发（例如：如果您在4核计算机上同时训练4个xgboost模型，请将每个模型进程固定到单独的核）。

使用Visual Studio：

与明威：

### 结论

使用Visual Studio而不使用CPU固定似乎是目前最好的选择。

对于希望获得大部分xgboost/LightGBM的超级用户的建议：

* 尽可能使用Visual Studio
* 无CPU固定的列车模型
* 并试图获得更高的CPU频率…

如果在Windows中被迫使用xgboost，则强制CPU固定以提高性能。

如果你有一个单一的模式来训练，GPU xgboost似乎是一条路要走，因为它今天变得多么稳定。你甚至不需要一个强大的服务器，甚至笔记本电脑的NVIDIA 1050 Ti比我们的怪物服务器性能更好。

好奇的是，在带有GPU xgboost的笔记本电脑上使用NVIDIA 1050 Ti（1.75ghz），训练一个模型需要92秒。这比最快的xgboost（Visual Studio+CPU pinning+9个物理内核）快28秒。一个过度锁定的工作站将把这段时间缩短到大约60秒。

Find below the **most brutal comparison in efficiency**, when using xgboost and CPU pinning:

Next part: [Investigating xgboost Exact scalability](https://medium.com/@Laurae2/investigating-xgboost-exact-scalability-d562b2b501c0)