# **Optiver低延时交易 核心解密（1）**

原创 愚夫 [愚夫一得](javascript:void(0);)

 2025年05月25日 11:12 上海

David Gross，在Optiver荷兰公司负责期权自动化交易系统的开发。在2022Cpp技术峰会上，他探讨了Optiver在C++低延迟自动化交易系统设计上的思考，并从以下4个主题进行了深入介绍：  
**数据模型**：数据是系统的基石，如何组织数据至关重要。  
**并发访问**：在高速交易环境中，如何处理并发数据  
**系统调优**：如何榨干硬件的每一滴性能？  
**性能测量**：如何量化系统的性能，找到瓶颈？

### 何时考虑性能问题？

Donald Knuth 有句名言：“过早优化是万恶之源。” 听起来很有道理。我的一个朋友也常说："必要时再优化"。我个人认同这种实用主义观点。

但昨天在CppCon大会上，Matya强调"实时系统设计初期就该考虑性能"。性能是设计出来，而不是改出来的。

这两派观点看似矛盾，但其实它们分别对应系统设计的战略与战术两个不同的层面：

* • **战略层面**，我们需要制定一个总体目标，比如 要求每秒处理百万事件或纳秒级响应，指导我们的大方向。
* • **战术层面**，我们需要考虑具体的实现细节，比如 “我应该在这个 std::vector 上调用 reserve 吗？”，“这个函数内联会更快吗？”，“哪个哈希表更节省空间？”等等。

Donald Knuth反对的是过早地在**战术**层面过度局部优化，这只会把代码搞复杂。而我强调的是在架构设计时在**战略**层面就要有性能意识。如果你一开始的系统设计（数据结构、并发模型）就有问题，后面再怎么在战术上进行局部优化，效果也可能微乎其微。今天的分享将兼顾这两个维度。

### 多少时延才算快？

在 CppCon 2017 上， Carl Cook 做了一个精彩的演讲，题目是《When a Microsecond Is an Eternity: High Performance Trading Systems in C++》。 他提到，一个优秀的软件交易系统，从接收行情数据到发出交易指令的端到端延迟，可以做到 **2.5 微秒** 左右。

如今市场已发生巨变，看看现在交易所（比如法兰克福的 Eurex）的数据。从接收到交易所发出行情数据（Trigger）到参与者订单发出（Target）的延迟，在过去几年里一直在下降，头部参与者已经逼近 **10 纳秒（ns）量级** 了！这比光在你我之间跑一个来回还快。

这种速度，常规的 CPU 和软件，甚至连最新的PCIe标准都几乎不可能做到。这是 FPGA的领域，用硬件描述语言（如 VHDL）在定制板卡上实现的逻辑。

### 低延迟 C++ vs FPGA ?

那么，C++ 在低延时游戏中已经过时了么？

我的答案是：C++ 不仅有一席之地，而且至关重要。原因如下：

* • **系统不止有“触发-响应”：** 一个完整的交易系统远比那个纳秒级的循环复杂。即使拥有能够每秒发送十万笔订单的FPGA，仍然需要配套的C++代码来接收消息并进行后续检查，例如风险控制、订单管理等。
* • **FPGA 很“单纯”：** FPGA 能做到极致的低延迟，但通常执行的是比特级的比较和操作等简单逻辑。对于衍生品定价或新闻内容理解等场景实现仍较复杂。
* • **权衡（Trade-off）：** 工程永远是关于权衡。FPGA 用硬件描述语言开发的方案非常昂贵且开发周期长，不如C++语言抽象和灵活。而且，不是所有策略都需要纳秒级的延迟。几微秒甚至几十微秒的延迟对于很多场景来说足够快，而 C++ 在这个领域依然是王者。

### 数据模型的重要性

我刚加入 Optiver 时遇到的一个故事：有一个算法 A，速度还行；后来一群聪明的工程师开发了更智能的算法 B，但 B 很慢。大家花了大量时间想让 B 变快，尝试了各种优化技巧（战术），但效果甚微。为什么？  
用 Profiler 分析一看，结果令人沮丧：虽然教科书上说80%的CPU时间通常消耗在20%的代码上，但当数据模型设计不当时，系统没有明显的性能瓶颈！CPU 时间均匀地分布在各处。

为什么会这样？ 因为这个算法的数据模型没有考虑到性能。数据散落在内存的各个角落，cache miss 严重，导致整个系统被数据访问拖慢了速度。

所以，低延迟设计的第一个核心原则就是：**关注数据，特别是数据在缓存中的布局和访问。**

你需要了解你的武器库：

* • L1 Cache：极快，但很小（比如 32KB）。
* • L2 Cache：较快，稍大（比如 1MB）。
* • L3 Cache：相对慢，但更大（比如 32MB），通常在多个核心间共享。

现代处理器的多级缓存架构要求我们在系统设计初期就充分考虑数据规模和访问模式。在交易系统中，我们可以预先估算需要处理的证券数量和数据结构大小。

另一个经常被忽视的因素是：在多核环境中运行时，应用程序会与其他进程共享最后级缓存。即使某些进程看似已经终止，其残留的缓存影响仍可能持续存在。

你的目标是让 CPU 访问的数据尽可能地命中 L1/L2 缓存。这意味着数据要：

* • **小而紧凑（Packed）：** 避免不必要的填充和浪费。
* • **局部性好（Locality）：** 一起访问的数据，物理上也要放在一起。

### 数据模型设计实例：合约容器

为了更具体地说明数据模型的重要性，我来分享一个关于**合约容器** 的设计。

在交易系统中，**Instrument** (合约) 是核心概念，比如股票、ETF、期权等等。一般来说，每个交易算法可能需要处理成百上千个合约。 算法需要根据 合约ID 不时地查询合约信息。

那么，我们应该如何设计 合约容器 呢？

最常用的可能是 std::unordered\_map<InstrumentId, Instrument>，查找快（O(1) 平均），而且 Instrument 的地址稳定（因为是节点存储）。但它的问题在于：**内存布局是随机分散的**。当你遍历或随机访问多个 Instrument 时，CPU 缓存会被大量换入换出，性能差。

在CppCon等会议关于性能优化的讨论中，常会看到std::list和std::vector的基准测试对比。需要明确的是，性能问题的本质不在于具体容器类型，而在于内存模型。std::list和std::unordered\_map都采用节点式内存分配，每个元素存储在独立的内存块中。

一个改进方案是使用 std::vector<Instrument>但这会破坏引用稳定性（插入/删除会使已有元素的引用失效）。

一个更好的折衷方案是 stable\_vector<Instrument>。你可以把它想象成一个 std::vector 套 std::static\_vector （或者更像 std::deque ），外面一层管理着指向内部数据块（Chunk）的指针，数据块内部是连续存储的。用一个向量 mChunks 来管理这些 chunk。 当需要访问元素时，通过 i / ChunkSize 和 i % ChunkSize 计算元素所在的 chunk 和 chunk 内的偏移量。你可以自定义数据块的大小（比如设置为 L2 缓存大小的倍数）。

stable\_vector 的优点：

* • 数据在 Chunk 内部是连续的，有一定**缓存局部性**。
* • 添加元素时，即使发生 Chunk 分配，已有元素的**地址（指针/引用）保持稳定**。
* • 查找需要计算两次索引（Chunk 索引 + 内部索引），比 vector 慢一点，但通常比 unordered\_map 的随机内存访问要快得多。这是为获得更好局部性所做的合理权衡。

这个简单的容器，却能带来显著的性能提升。 关键在于它保证了数据的连续存储和指针的稳定性，从而提高了 cache 命中率，降低了内存访问延迟。

### 如何评估数据局部性？

评估数据局部性不是看它占用了多少内存（RSS），关键指标是看它实际**访问**了多少内存，也就是**工作集大小（Working Set Size, WSS）**。

Netflix性能工程大牛Brendan Gregg 有工具可以估算 WSS（通过统计访问过的内存页）。虽然页与缓存行的粒度不同，但该工具仍能有效反映数据局部性问题。

一个有趣的测试：比较 stable\_vector<int,4\*4096>和 std::unordered\_map<int,int>的WSS占用，都存储 10 万个元素，然后遍历求和。

**特别注意**：在std::unordered\_map测试中插入std::list操作以打乱堆分配模式，避免微基准测试产生误导性结果（实际生产环境中内存分配是随机的）。

结果显示，unordered\_map的 WSS（Ref(MB) ≈ 391MB）远大于 stable\_vector （Ref(MB) ≈ 1MB），尽管它们的 RSS（总占用内存）差不多（约 3GB）。这印证了 unordered\_map存在严重的页级局部性问题，将导致大量TLB缺失。

**关键启示：** 选择数据结构时，不仅要考虑算法复杂度，更要思考它在内存中的布局以及如何与缓存交互。

未完，待续。

点击**关注**，共同进步

卡通画

AI 生成的内容可能不正确。

**愚夫一得**

某交易所技术人。在持续学习中和你聊聊交易、研发、运维、数字化、管理那些事。

135篇原创内容

公众号

#### 参考文档

*[1] 讲座视频：https://www.youtube.com/watch?v=8uAW5FQtcvE*