**并行模拟框架优化记录**

**一、消息映射：从固定映射到随机映射**

框架调度的对象是系统中所有组件产生的消息。这些组件包括core组件、memory组件、cache组件和mesh组件，共计三千余个。旧框架将这些组件按照组件对象的ID进行成几组，每组的消息分发给一个工作线程处理。这种映射方案常常导致不同工作线程的工作量严重不均衡，因为：一、每个组件对象的ID由其组件的类型及在拓扑中位置决定，这导致实现中会将同一类组件的连续多个对象映射到同一个工作线程上。二、不同类型组件，其消息数量、延迟等属性特征差别较大。由于负载分配不均，多线程优势无法得到充分发挥。

新的框架采用随机映射的方式，不再按照拓扑结构分配消息，每个线程可能执行到任何一个组件的消息，每个组件的消息也可能会在任何一个工作线程上被处理。由组件类型差异而导致的工作线程负载不均衡问题便不复存在了。

**二、消息队列：从全局互斥锁到无锁队列**

各个工作线程与主线程之间通过消息队列进行通信。工作线程从消息队列中取出一组要在本回合处理的消息，并在处理完成后，带回新产生的消息。初始的机制是使用一个全局的消息队列，并使用一个互斥锁隔离各个线程的消息存取。这种大锁结构带来的问题是：在消息粒度较小而使得消息存取较频繁的情况下，多个线程将会频繁地申请该锁，存取消息的过程将变得严重串行化，成为消息调度功能的瓶颈。

优化后，框架为每个主线程—工作线程对的每个方向（主线程转发消息到工作线程以及工作线程转发消息到主线程）提供了一个循环消息队列，基于单一读者单一写者模型，可以不使用锁实现每个线程对消息的同步。这大大降低了消息的存取开销，提高了框架的并行性。

**三、模拟进度控制：从current\_time-safe\_time模型到cycle-by-cycle模型**

框架的模拟算法基于并行离散事件模拟（PDES）模型，其实现分为乐观方式和悲观方式。这两种方式是在模拟速度与模拟精度上的不同权衡。乐观的PDES新老框架牺牲了一部分精度以换取更快的模拟速度，需要配套实现回滚机制。由于目标模拟器对较高精度的要求，实现中选择了悲观的PDES算法，在保证精确性的基础上实现并行模拟。

优化之初，对模拟进度的控制采用的是current\_time-safe\_time模型。每个组件的对象有两个标志该对象运行进度的变量：current\_time和safe\_time。current\_time指示该对象下一个要执行的消息的接收时间戳不会小于该值。而safe\_time指示该对象不会再收到接收时间戳小于该值的消息。由于：1）一个对象的safe\_time由其所有上游对象的current\_time与消息延迟计算得到。2）一个对象的current\_time取值为该对象最近执行的一个消息的接收时间戳，并且当该对象的消息队列中不存在介于其current\_time与safe\_time之间的消息时，该对象的current\_time直接推进到safe\_time。基于以上两点更新所有对象的current\_time和safe\_time是一个循环迭代的过程。在对象数目较多且上下游关系形成若干长链或者长环的情况下，迭代的次数将快速增加，以至于成为了模拟框架的性能瓶颈。这也正是我们所面对的情景。

优化后，在主线程的统一调度下，每个线程一个cycle一个cycle地执行，假设某个cycle有1000条消息需要处理，主线程会先给每个模拟线程发10个，然后实时监控处理情况，给处理快的再发一些，以平衡负载。当这1000条消息都处理完后，再进入下一个cycle的处理。

**模拟框架性能评估**

**横坐标为线程数，纵坐标为“运行时间” 横坐标为线程数**

对于使用多程序方式运行的模拟器，如上左图所示，模拟框架使用的线程数从1增加至7时，三个测试程序都有着3倍以上的加速效果。以test-math测试程序为例，随着框架线程数增加，系统运行时间从1635秒逐渐降至487秒，加速比超过3.36。

对于使用多线程方式运行的模拟器，如上右图所示，使用框架线程数增加并不会带来整体运行时间的。分析其运行时间的组成，可以发现，随着线程数的增加，框架等待消息到来所花费时间的比例随着线程数增加而逐渐从30%提高的80%以上，程序的实际执行上层模拟任务的时间从70%降低到20%以下。

由前面两幅图的对比，我们发现框架对上层模拟器的加速效果取决于模拟器中消息的密度。

1. 当消息密度较高、框架有较为持续的供给时，可以达到3倍以上的加速效果。如图中以多程序方式运行test-math、test-add和test-sort的模拟器，其消息分布的绝大部分情况都是每个cycle有400个以上的组建对象接收到消息。
2. 当消息密度较低、框架无法获得较为持续的供给时，框架的加速效果受到限制。我们通过添加barrier，提升了模拟器中消息的密度，框架的加速效果逐渐体现了出来。以多线程方式运行kmp程序的模拟器为例，在添加barrier提升消息密度之前，见图中kmp\_no\_barrier，模拟器任意一个cycle内会收到消息的组件对象不超过100个(放大该部分可发现，事实上每个 cycle收到消息的组件对象数量连20个都不到)。如图2所示，这中情况下几乎达不到任何加速的效果。**添加barrier后，见图中kmp\_barrier\_full-stage和kmp\_barrier\_stage1-2，模拟器中消息的密度较为均匀地分布在各个取值区间内，模拟器的加速能力也得到了适当的发挥**。

**不同用例（多任务/多线程）及不同运行方式（有无barrier）下平均每个cycle内消息并比例**

若果将框架视为邮局，框架内的工作线程视为邮递员，而模拟器中的各个组件间的消息传递看作人之间的信件往来。在信件交流频率很低的情况下，加倍地雇佣邮递员是无法达到显著加快信件邮递速度的目标的，而只会造成资源（CPU时间）浪费、机构（框架）臃肿、人（线程）浮于事。