一篇文章入门仿真软件性能优化

原创 邓子平 多物理场仿真技术

之前有讨论过关于支付宝和工业仿真软件数据特点的对比，工业仿真软件慢的根本原因在于模型的整体性和难以拆分。

根据经验，笔者将加速仿真软件性能方法，归纳为四个方面：

1. 分治(Divide And Conquer)

2. 单一数据模型(Single Data Model)

3. 优化高频操作

(Optimize High-frequency Operation)

4. 改进编程语言使用和底层算法

(Improve Language Usage and Basic Algorithm)

1. 分治(Divide And Conquer)

1.1.分治是最容易理解的，从最简单的文件读写说起。在有限元模型中，通常我们会把网格文件保存在一个文件中。文件会包含节点，单元，边界荷载材料等所有信息。把所有信息放在一个文件的好处是容易处理，但是当文件规模达到G的级别时，比如10G，读写就会出现性能瓶颈。网格文件中通常节点和单元会占据绝大部分数据，合理的做法就是分出两个进程，一个读节点，一个读单元；进一步可以将节点单元放在不同文件中。在后处理文件中，存在众多结果数据，对于10G网格的模型，后处理器根本无法一次性装下，分开不同文件处理是必然选择。现在固态硬盘的读写速率普遍达到500M/秒以上，硬件I/O已经不是瓶颈，需要改进的是软件策略。

1.2.有朋友会说，这个分治操作太简单了。是的，因为网格数据节点和单元本身就是分开的，我们只是在文件级别进行分治，进一步比较难的是在模型数据层面进行分治。目前（2020年）大部分CAD或仿真软件都很难导入一个1G的step文件，原因就在于Step文件是采用的类似索引结构存储信息，无法并行；为了提升性能，可以写一个转换器，将其按照某种属性分类，分别保存为文件，再次读入的时候就容易并行化，特别适用于需要反复读入的情况。

1.3.在仿真软件中，数据类型主要分为业务数据，几何模型数据，渲染数据，网格数据，求解器数据，结果数据。分别说一下这些数据的特点：业务数据多为属性或者参数，控制等数据，特点是数据量小；几何数据大多用BRep或者参数表示；渲染数据多为点数据和三角形，网格数据为节点和单元，荷载，边界，工况等；求解器数据为各种大规模矩阵；结果数据在网格数据基础上，给节点或单元附上各种属性数据，或者对原始数据进行二次提取和加工。

对同一个模型大小的估计，不同数据类型和表达方式可以得出不同结论。比如计算材料学中的一个晶胞，是一个长方体，业务数据可能就只有一个点坐标；对于参数几何而言就是简单的长宽高；对渲染而言是12个三角面片(每个面两个三角形)；对网格而言，如果网格划分的很细，可能是百万的三角形；对于求解器而言，高阶单元在网格的基础上，还要乘上好几倍；对于仿真来说，模型大小主要由最终求解器数据的矩阵规模来确定，矩阵的规模也就是要求的自由度的规模。

使用分治方法之前，需要明确对象是否合适分开处理，如果能满足以下条件就可以采用：

1. 数据之间耦合度低；

2. 分开之后的数据保留各自的属性，尽量保留整体属性；

3. 分治的相关数据操作之后能再次组合在一起；

4. 分治的各个数据规模开销相当

理清了数据的特点和分治的要求之后，我们会发现分治的核心是将模型数据分离，但是这与我们之前提到的仿真模型是整体的概念相互矛盾，这就是仿真模型加速的难点所在：既要分治，又要保持整体性！

几何数据中，我们尽量在设计阶段将几何分开设计，比如四条相同腿的桌子，桌面和腿分开设计，不要进行布尔运算，对于腿，不要使用深度拷贝操作，尽可能使用类似实例操作，也就是创建一根腿，其它三个复制原始几何+变换矩阵，这样可以省下建模和离散渲染的开销，在数据量多的时候大幅提升性能。对原始几何数据根据属性进行分类，提取特征，在需要全局进行数据计算时首先利用属性特征进行检索。最典型的例子是在全局进行干涉检查，如果任意两个对象两两检查，算法复杂度为N的平方，数据量大的时候行不通，而通过类似Octree, kbtree的结构，将所有对象分开处理，只检索对象附近的区域，会大大提升性能。这是利用空间树实现分治的一种方法，实际中可以根据业务属性分组，也就是利用分治的办法提升性能。

在求解器中，最终我们需要求解大规模线性方程组，之前介绍的Krylov子空间方法也是比较典型的分治法，即将矩阵进行降阶，分成维度更低的矩阵，再进行求和运算；而用于快速求解满秩矩阵的快速多级，本质上也是分治法，将矩阵分成不同的区域，先对不同区域内进行计算，再在区域外计算，最后组合结果，快速多级将满秩矩阵的求解复杂度保持在了线性水平。求解器的分治算法是分治方法中最难的部分，也是属于比较底层的算法。

将模型数据进行分治之后，剩下的操作就相对容易了，主要通过多线程，多进程，群组服务器，而这些操作可以自己开发，也可以利用已有的MPI工具，如OpenMP，OpenMPI。需要提一下GPU计

算，在渲染数据和矩阵计算中，可以利用GPU来进行，而非CPU，目前主流仿真软件都支持GPU计算，这个之前也有专门的相关介绍，参考附录。

2. 单一数据模型(Single Data Model)

这个也比较容易理解，对于大模型，始终只保留一份数据，避免大数据的复制拷贝等操作，最好从制度上保证这种现象出现，比如核心数据只能使用指针，或者智能指针；对于临时出现的大数据，用完即销毁；对于海量小数据，自动分配大内存池和活动空间

单一数据模型说起来很容易，但实际中很难做到，主要还是对大模型的操作比较繁琐，也难调试

3. 优化高频操作

(Optimize High-frequency operation)

高频操作也是影响性能的一个重要因素。以求解器数据更新操作举例，比如一个单元被修改，常规做法是会立马通知并更新矩阵，如果修改了十个单元，那就会通知十次，对于小模型，这种操作很难觉察到性能问题，但如果是修改百万的数据，这种性能影响就非常明显了；理想的做法是修改数据后不要马上更新，而是等到需要使用数据计算之前再去统一更新。

控制高频操作的典型方法是使用缓存方法和之前介绍设计模式提到的Observer模式。即将需要频繁更新的数据进行缓存，只对缓存数据进行操作，通过Observer在适当的时候进行更新，避免全局进行频繁操作。高频操作无论是在几何，网格，还是求解器都是影响性能的一大瓶颈，而且是完全可以通过简单的技术手段避免的。

此外通过数据分级也是有效解决高频操作的方法，即根据业务逻辑，尽可能采用金字塔层级而非扁平的数据模型，这种结构有利于从架构上实现最小权力原则和高频局部数据操作，提升性能。

4. 改进编程语言使用和算法

(Improve Language Usage and Basic

Algorithm)

如果做好了之前的三点，那剩下的就是需要从语言和底层算法方面改进了，比如C++中合理使用vector，map，unorder_map；避免多重循环，慎重选择复数类等等，这个需要的是长期开发积累。

此外，一些基础算法也要重视，比如自己开发面面求交算法，性能，稳定性，准确性要非常好的拿捏，以符合业务需求，用户可能只需要判断是否相交，而不需要求交线，不求交线可以提升性能，对于不可避免的高频计算，可以节省不少开销。

其实关于工业仿真软件性能的话题早有探讨，只是没有系统地分析。工业仿真软件追求的不是一种功能，黑科技，或先进的技术手段，看重的是准确性，稳定性，可靠性和性能。当然这几种指标并不是独立的，比如加密网格会提高准确性，但会降低性能，一般来说准确性是优先考虑的；检查各项数据的完整性和正确性可以提高稳定性和可靠性，但无疑也要消耗更多资源；所以在实际研发中这些指标有可能相互排斥，需要根据具体的业务来选择相应的策略。

工业软件的性能问题是一个可以一直讨论的话题，后续还将结合实际做更多探讨。

一篇文章入门求解器模型降阶方法

FEM之在求解器中使用设计模式(8)---Command和Observer模式

一篇文章入门HPC(高性能计算)

FEM之求解器加速(1)---HPC简介