arXiv:2603.04829v1 [physics.comp-ph] 5 Mar 2026

用于邻域计算的树代码与排序扫描算法：面向缓存的比较

Dominik Krengel[a,*]、Yuki Watanabe[b]、Ko Kandori[b]、Jian Chen[c]、Hans-Georg Matuttis[b]

[a]东京海洋科技大学海洋资源与能源系，日本神奈川县横滨市金泽区昭和町3173-25，邮编236-0001[b]电气通信大学机械与智能系统工程系，日本东京都调布市调布丘1-5-1，邮编182-8585[c]日本海洋研究开发机构数学科学与先进技术中心，日本神奈川县横滨市金泽区昭和町3173-25，邮编236-0001

摘要

邻域算法在离散元方法（DEM）中可能占用相当比例的计算机运行时间。排序扫描算法在某些方面很理想，因为它只处理相对位置在一个坐标方向上发生变化的粒子，但所有粒子的其他方向也必须处理。相比之下，树代码只处理相邻颗粒。我们比较了用于旋转滚筒内多边形颗粒二维离散元模拟的排序扫描与树编码邻域算法，模拟粒子数最多达 12000 个。我们讨论了系统规模和内联对缓存性能的影响。树代码的性能略好，但圈复杂度显著增加。尤其是，它的一项优势是改善了共享内存并行化的可能性。

关键词：离散元法、邻域计算、树代码、排序与扫描、性能优化

1. 引言

对于离散元法（DEM），需要计算所有接触粒子之间的相互作用。为了高效地评估接触粒子之间的短程接触力，并跳过非接触粒子的计算，人们设计了多种不同的邻域算法（综述见参考文献 [1]）。某些邻域算法（如 Verlet 表或链式单元法）必须在每个时间步为所有粒子从头重建接触列表，或者在经过一定数量的时间步后重建列表时，对粒子的最大运动范围作出有风险的假设。理想情况下，系统中的运动越少，所需计算也应越少；对于致密系统尤其如此，因为相邻颗粒会限制运动范围，时间积分器允许的最大步长也会对其加以限制。理想情况下，接触列表（或邻域列表，即基于几何条件可能发生接触的粒子索引对）在离散元模拟中，只有相对位置发生变化后，才更新接触列表（或邻域列表）。对于包围盒（即极值坐标），支持这种面向更新的实现的算法包括排序扫描算法 [1] 以及树代码 [2, 3], ；后者将在第 2 节和第 3 节中分别作进一步介绍。

只更新数据结构中的变化，同时将未变化的相互作用对保留在接触列表中，不仅能提高邻域计算的效率，与从头重建邻域相比还有另一项优势：除了粒子索引，还可以存储“负载”，即有关接触粒子的额外信息，例如上一时间步的摩擦值（这是 Cundall–Strack 摩擦 [4] 所需的），或者其他会随粘着时间变化的滞回摩擦力 [5]的首次接触时间。对于三维非球形粒子的仿真，计算开销主要集中在重叠计算 [6], 上；而对于球形粒子或二维模拟，邻域算法则可能成为限制性能的因素。

*通讯作者电子邮箱：
dominik.krengel@kaiyodai.ac.jp（Dominik Krengel）

从头构建含 N 个对象的排序与扫描算法和树代码，其计算复杂度均为 O(N log N)。这两种算法都能受益于使用轴对齐包围盒，即极值坐标，而非采用颗粒轮廓的完整几何信息。更新包围盒时，只有粒子的相对位置发生变化，才会影响邻域列表，而这种粒子只占实际粒子的一部分。更新包围盒并核验粒子相对位置的计算开销为 O(N)，但重新排列与变化的邻域相关的信息所需的运算次数远少于 N。为验证理论上的整体 O(N) 复杂度，我们将分析采用排序与扫描算法及树代码实现的离散元模拟的性能。

计算机科学中的传统复杂度分析侧重于必要运算的数量，而忽略了在内存与中央处理器之间传输必要数据所需的时间。在基于缓存的处理器架构中，每当使用的数据量达到缓存大小时，性能就会因缓存未命中以及从下一级更大、速度更慢的内存单元重新加载数据而下降，因此实际性能可能偏离理论预测。这种影响会显著降低整体性能，但通常不会反映在“缓存无关”的复杂度分析中。在线性代数领域，关于缓存效应的开创性研究是通过对采用不同内核的向量化矩阵乘法进行基准测试完成的；结果显示，对于不理想的算法和矩阵规模，中央处理器时间可能比理论上的 $O(N^3)$ 增加一到两个数量级 [7]。缓存未命中次数的理论评估十分繁琐，并且取决于硬件细节；这些细节会随总线变化，也就是说，会因主板不同而异。尽管如此，要依据实际实现的性能分析判断排序与扫描算法和树代码是否适用，仍然需要具备缓存感知的算法实践知识。

本文探讨了二维情况下，排序与扫描实现相较于树代码的优劣。我们采用一种多边形离散元代码 [1]，其中的粒子略微细长；这类粒子应能代表尺寸大致相同的任何（近乎）不可穿透的离散元颗粒，见图1。为了解实现细节的影响，我们对排序扫描算法和树代码都进行了基准测试。在缓存大小不同的机器上测量这两种算法的相对性能（见表1）。计时的主要部分针对 MATLAB 解释器代码。此外，我们通过“MATLAB Coder”将代码转换并编译为 C代码，再进行基准测试，以考察编译型语言代码的性能表现。我们还考察函数内联的影响，并以圈复杂度 [8, 9] 分析算法复杂度。本研究中的基准测试和分析有助于理解并改进离散元模拟中的邻域检测性能。对于具有接触相互作用的“固体”粒子，基于包围盒更新的排序与扫描算法和树代码都非常高效。尽管如此，这些代码也有缺点：当重叠较大、存在长程力或相互作用数量较多时，由于邻域变化过多，每个时间步需要执行过多更新运算，因此效率可能较低。这类系统的例子包括光滑粒子流体动力学（SPH）[10] 和移动粒子半隐式法（MPS）[11]等方法中的代表性质量。出于同样的原因，本文讨论的算法在含有额外或专有长程力的仿真中也可能效率较低，例如黏土离散元模拟中的范德华力 [12]，其截断势作用范围过长；此外还包括引力和静电库仑力。

2. 排序与扫描邻域计算

Baraff [13] 提出了用于邻域计算的“排序与扫描”算法 [14]，后来该算法被称为“排序剔除”。粒子（其极值坐标称为“包围盒”）的坐标与包围盒的所有权索引一起按顺序排列在列表中。每当某个包围盒的底部坐标移到另一个粒子的顶部包围盒之下时，就可能发生碰撞。图 2 展示了三个包围盒。从时间步 n 到时间步 $n+1$，粒子 1 和 2 开始发生重叠。首先，按照时间步 n 排序的列表中各包围盒的位置会更新为新位置。在重排序过程中，发现粒子 1 的顶部坐标 t_1 和粒子 2 的底部坐标 b_2 需要交换，这意味着粒子 1 和 2 之间可能发生碰撞。对两个维度使用同一算法时，彼此斜向移动的粒子可能会在邻域列表中被重复录入。可通过额外使用旧包围盒列表来避免这一情况。沿 x坐标方向出现的新重叠总会被录入；而沿 y轴方向的重叠，只有在此前 x方向已发生重叠时才会被录入。排序时可采用成对交换（“冒泡排序”），因为列表已经部分有序。

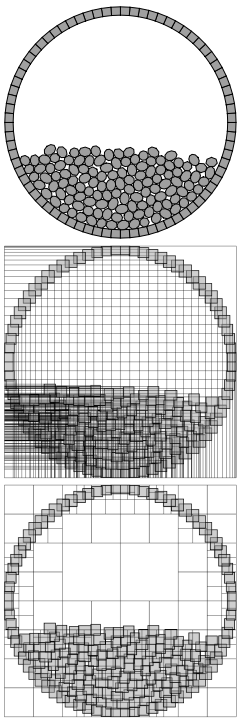


图 1：粒子数量较少的缩小后的鼓（上）；排序与扫描算法仅使用 x 轴和 y 轴方向上的上边界包围盒坐标，这些坐标的数量超过可能发生的粒子接触数（中）；相比之下，树代码中的包围盒基于系统划分（下）。

初始化时，若排序顺序不理想，可能会导致大量 $O(N^2)$ 次操作。初始化时，我们可以先录入质心而非包围盒，再用快速排序等更快的排序算法对其排序。随后，将质心替换为包围盒，并对该列表执行增量排序，以获得初始接触列表。释放接触时，应使用另一个循环，剔除所有包围盒不再重叠的粒子对。

3. 用于邻域计算的树代码

与排序-扫描方法的情况类似，使用树代码处理包围盒，而非详细的粒子几何形状，也是合理的。每个时间步中，位置更新后都需要灵活地重构树，这比通过重排序更新包围盒列表更为复杂。若算法每个时间步都从头构建树结构，耗费 N log N，则无法与只更新列表并处理变化的排序-扫描方法竞争。因此，

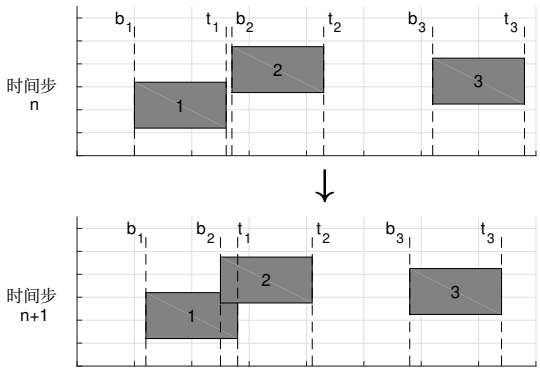


图 2：排序扫描算法的有效步骤：从时间步 n 到时间步 n + 1，包围盒坐标 b_2 和 t_1 沿 x 轴改变顺序，因此必须将粒子对 1 和 2 记录到邻域列表中。

我们设计了一种算法，每个时间步只更新现有的树结构。对于二维（四叉）树，不同于二叉树中的左、右两个方向，我们需要处理四个方向：东北（NE）、西北（NW）、东南（SE）和西南（SW），如图 3（左上）所示。将一个区域划分为四个对称的象限即可得到子节点（右上）；原较大区域对应的旧节点则成为父节点。

接下来，绘出粒子占据情况后，可以看到，单元划分和树的簿记有两种可能的方式：“最小树”在树的所有层级上都采用尽可能大的单元，因此相邻单元的大小可能不同（图 3 左下）。另一种方法（Vemuri 等人 [2, 3], 在三维情形下采用的方法）则只分配大小相同的最小单元，并将它们全部置于树的最低“叶”层（图 3 右下）。在这种情况下，关键在于对存储的树结构进行“剪枝”，从而只需管理已占据的分支。后文将会说明，我们的“最小树”可以在线性时间内搜索邻居，因为树结构中包含了相邻已占据单元的信息。相比之下，Vemuri 方法需要 N log N 次运算，因为对于所有 N 个单元，相邻单元都可能连接到树的 log N 个较高层级中的任意一层的树分支。本文主要讨论尺寸分散度较小且伸长率较低的粒子，见图 1。壁面粒子和其他大型粒子会进一步划分为若干至多个包围盒，其处理方法将在第 3.4 节说明。

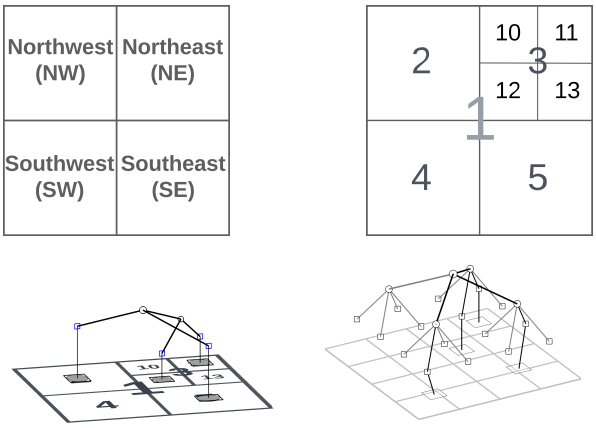


图3：四叉树中的方向（左上）、带有已占用单元的单元编号示例（右上），以及针对相同占用情况的我们的“最小树”（左下），与Vemuri 等人 [2, 3] 的“剪枝树”（右下）形成对比；其中仅使用黑色分支，而灰色分支可能在粒子移动时被激活。

3.1. 树代码的历史

“树”这一术语由 Cayley 于 19 世纪引入数学领域，而树形数据结构在计算机科学中的应用自20世纪60年代起逐渐兴起。Finkel 和 Bentley [15] 提出了二维结构的“四叉树”和三维结构的“八叉树”这两个术语。二叉树（一维）、四叉树和八叉树在固定边界上运作，而 k-d树 [16],则允许边界发生变化。20世纪80年代 [17]，树结构首次被引入物理模拟，用于降低引力产生的“长程”相互作用的计算复杂度 $O(N^2)$。这类模拟并未使用树进行邻域计算，因为无论如何所有粒子都会相互作用；树结构用于连续求和力，并利用局部获得的质心截断力的计算 [18]。在刚体固体建模领域，M. C. Lin [19]在多面体的最近特征算法中使用了树代码，并提出将树代码用于邻域算法，但并未实现。Vemuri 等人 [2, 3] 在三维中实现了一种算法，其更新所需时间为 O(N)，但对于单分散球形颗粒，邻域查找的最坏情况复杂度为 N log(N)。我们稍后将把这一方法与我们针对低多分散性、采用不同空间划分的方法进行比较。Wackenhut [20] 为多分散圆形颗粒实现了树代码，但没有可查阅的实现细节、运行时间或后续发表的论文。Schwartz 等人 [21] 在软球离散元法中使用了一种高性能并行引力树代码（最初由Stadel [22] 和Richardson [23]为天体物理学应用而设计）以N log(N)时间生成接触列表，但没有更多算法细节可供参考。在离散元法中树代码的背景下，Duriez 等人 [24] 将树结构用于离散元法粒子，其中相互作用计算通过水平集来定义。

3.2. 一维中的二叉树

通过树代码进行邻域计算的要点，最好先在一维情况下加以说明。首先，空间划分会被反复二等分，直到每个划分中只包含一个质心（图 4 中的“Before”）。相邻划分中的粒子（图 4 中“Tree before updates”的“叶节点”中的粒子）会被录入接触列表，作为潜在的碰撞伙伴。（如果根据几何条件它们不可能发生相互作用，则在基于其不重叠包围盒的重叠计算中会跳过这些粒子。）在图 4 中，包含粒子编号 i1 . . . i8 的叶节点以彩色绘制，父节点则以灰色表示。数组元素的索引（尽管也可以使用指针构建树结构，但在我们的方法中使用的是基于列表的树）以括号中的黑色数字对（m, n）表示，这些数字给出了存储基于列表的树的节点的位置，其中 m 表示层级（m = 1 是根节点，即树的最顶层节点），n 表示树中被占用的节点。树结构通过递归划分系统获得，直到每个叶节点只包含一个包围盒。红色节点是叶节点，在单个单元中仅分配有一个包围盒。在我们的方法中，叶节点表示大小不一的单元。父节点不“拥有”任何粒子（包围盒）。带有黄色和橙色边框的节点是在节点划分过程中必须新建的节点，因为在节点分裂期间，曾暂时有两个或更多包围盒分配给这些节点。在更新之前，我们有八个粒子 i1 到 i8，它们位于树中各分支底部（最低端）的叶节点内，处于大小不同的单元中的不同位置。接下来，i5（绿色）在物理上向“东”移动，越过系统中心，进入当前由 i6 占据的单元。因此，在步骤 1 中，i5 必须在树结构中向上移动到适合其新位置的最高节点。在图 4 的步骤 2 中，粒子 5 已开始沿树向下、朝叶节点方向移动。在步骤 3 中，当它遇到已被 i6 占据的节点时，必须在下一层新建两个叶节点，同时减小单元大小。在我们的方法中，一维情况下的单元大小各不相同，并且所有单元都被占用；而 Vemuri 等人 [2, 3], 的方法则 .....

单元的大小会相同，但其中一些单元不会被占用。

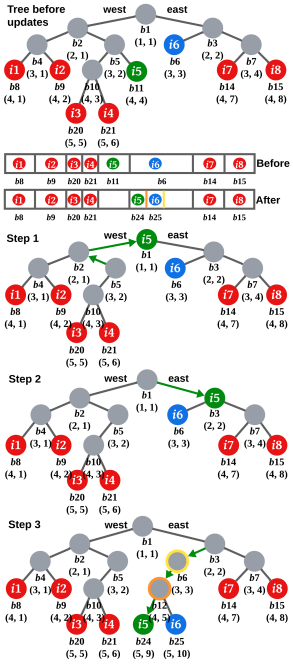


图4：一维树代码示例。顶部为原始树结构，其下依次展示粒子 i5 移动前后单元的占用情况和大小。接下来的三棵树展示粒子上升至最高节点（步骤1），然后向下遍历（步骤2），直至遇到一个已占用的节点，最后在较低层创建两个叶节点（步骤3）。

3.3. 二维树代码的实现

图 5 展示了二维四叉树的程序流程：首先，针对等尺寸粒子对树进行初始化和更新，从而可以初始化和更新邻域包围盒对列表。树结构的初始化包括两个部分：第一部分是构建树结构，即对系统进行规则划分，以创建由分割线构成的坐标数组。随后，根据分割线分配包围盒。然后，将包围盒分配给彼此独立的新子节点，这可能需要向树中添加新节点。第二部分是构建相邻包围盒对的列表，该列表基于节点的相对位置以及分配给这些节点的包围盒。完整的树仅在开始时构建，之后在更新过程中只执行增量更改，这减少了计算量，因为相对位置未发生变化的粒子无需进行处理。

仿真过程中的计算开销仅取决于更新所需的时间，而初始化过程在程序开始时只执行一次，因此其计算成本可以忽略不计。

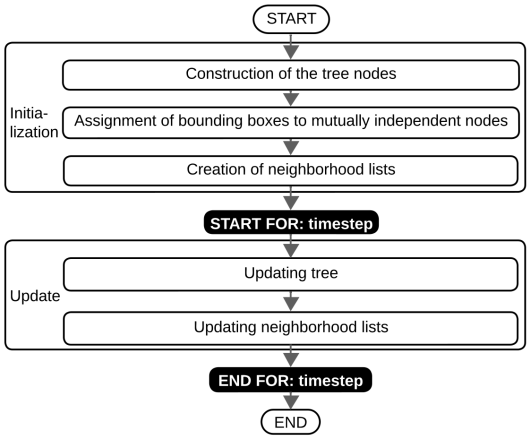


图5：二维四叉树邻域算法的简化流程图。

对于二维四叉树，其子节点的划分方式与一维二叉树类似。不同之处在于，四叉树可能包含未分配包围盒的空叶节点。粒子获得新位置后，图6中的树更新过程与一维情形相同：粒子先上移至根据其 xy 坐标可确定的最高父节点，然后根据坐标指示的方向沿树向下移动。在最低层，粒子要么被分配到已有的空叶节点；要么，如果相应位置上的叶节点已被占用，就将该叶节点拆分，并把两个粒子分配到下方一层的新叶节点中。更新过程本质上是顺序执行的，至少在当前程序版本中如此；但如第4节所述，其开销低于排序扫描算法的更新开销。

在一维情况下，只有沿一个方向（朝东和西）的相邻单元中的粒子，而在二维情况下，则必须处理四个方向（图 3 左上）。构建接触列表时遍历树，也必须考虑空节点的存在。对于一维树，如果某个方向上的粒子距离过远，就可以终止搜索碰撞伙伴；而在二维情况下，可能的相互作用伙伴或许存储在一个空单元之外的单元中。此外，角落处还可能有距离较近的粒子

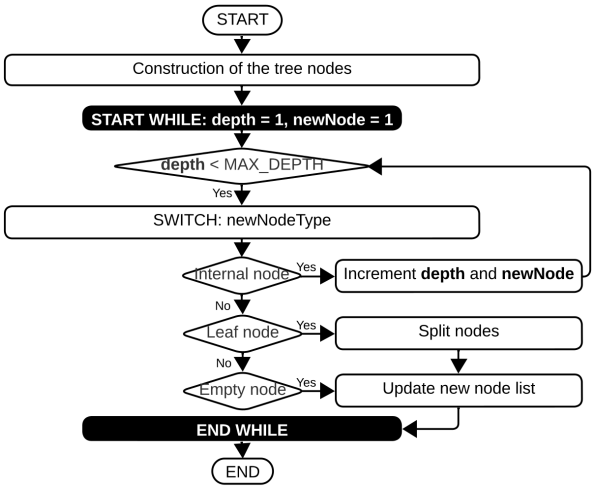


图6：四叉树更新流程图

这些粒子有三个可能的相邻单元。因此，需要递归处理才能找到所有相邻节点。此外，由于我们处理的是细长边界框，扩展到二维后，可能的邻域不仅要延伸到最近邻，还要延伸到次近邻。因此，从一维到二维，算法会发生显著变化，不仅涉及树的构建和更新，也涉及邻域列表的构建。

3.4. 大型粒子和壁面粒子

到目前为止，我们只讨论了尺寸大致相同的粒子。在许多离散元模拟中，需要跨越整个系统规模的壁面。用较小粒子拼成直壁会在粒子接合处引入额外法向，这可能导致表面实际上凹凸不平。Krijgsman 等人基于树代码在较大尺寸分散度情况下可能不够高效的论断，开发了一种采用层次网格结构的邻域算法：“用于接触检测的树数据结构不允许在层次结构的每一层独立选择单元格大小，因此无法针对各种粒子尺寸分布进行优化 [. . .]” [25]。此外，访问树中的相邻子立方体并不直接，因为可能存在来自不同树分支的节点；由于该方法此后不再使用，此处不再详述。相比之下，我们的方法允许将较大粒子（尤其是跨越系统整个高度或宽度的壁面）划分为（部分重叠的）包围盒，其大小与颗粒粒子相当，如图7所示。因此，尺寸分散度本身对我们这类实现而言并非问题。

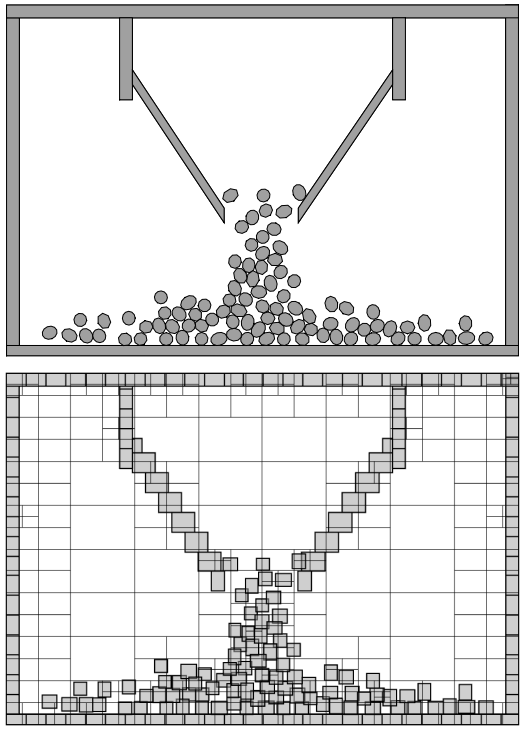


图7：实际的离散元法配置（上）以及树代码对应的不同大小的包围盒（部分包含粒子，部分为空）（下）。

4. 排序扫描法与四叉树的性能比较

我们使用 MATLAB 实现并比较了这两种邻域算法的性能。MATLAB 是一种解释型语言，但会使用部分编译的函数来消除开销。此外，它支持内联，因此可以重现编译型语言的典型特性。仿真主要采用专著 [1], 中的二维多边形离散元代码，但使用了 Krengel 和 Matuttis [26]的摩擦实现。我们处理的都是轴对齐包围盒，而非定向边界框；所有涉及斜向接触的计算任务均交由离散元法的重叠计算处理。我们使用表1中的机器，评估了从 1000 到超过 10000 个包围盒时的性能。如图8所示，尽管 Xeon56 的时钟频率较低，但其程序运行速度更快，且配备 DDR4 内存，因此我们使用该机器进行后续性能评估。

表1中的核心数与本文讨论的邻域例程的计时无关，因为与力计算相比，这些例程的计算开销很小。不过，为了加快仿真本身的速度，我们对力计算进行了并行化，因为对于最大的系统，即使使用单核心执行，最大的仿真仍需约一周才能完成。

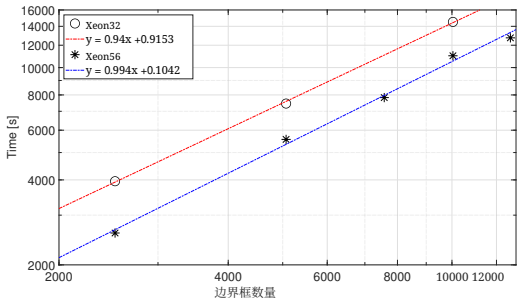


图 8：树代码在我们的计算服务器 Xeon32（3.3 GHz、512KB L1缓存和 DDR3内存）与 Xeon56（2.4 GHz、896KB L1缓存）上的执行时间比较。尽管时钟频率较低，Xeon56 仍然更快，这是因为其内存技术更先进。

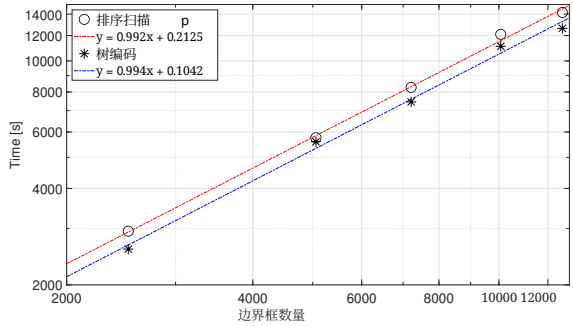


图9：在 Xeon56 上执行 DEM 仿真时邻域部分的耗时。耗时与粒子数量呈线性关系，即复杂度为 O(N)。在各种系统规模下，树代码的耗时仅约为排序扫描算法的 90%。

在研究过程中，面向更高性能的 Apple Silicon 处理器已可用于 Mac mini，因此我们将它们加入了比较。与英特尔处理器相比，它们同时以不同的时钟频率运行不同的核心：P核心（“性能”）的时钟频率较高，能效核心（“经济”）的时钟频率较低，见表1。与 Xeon处理器相比，Apple Silicon 的一级缓存更小，而 P核心的二级缓存更大，但能效核心并非如此。Apple Silicon 上的三级缓存似乎存在，但没有可用的规格说明。

4.1. CPU总时间

对于图1所示的滚筒系统，两种算法的复杂度均为 O(N)，但树代码所需的 CPU时间约为排序-扫描方法的 90%，见图9。诚然，旋转滚筒让树代码占据优势，因为粒子沿两个维度的包围盒坐标会不断变化，即使这些粒子之间不可能发生任何相互作用也是如此。对于这类系统，四叉树邻域算法更胜一筹。此外，树更新所需的时间仅为排序与扫描更新的十分之一，见图10。这意味着树代码中 80% 的 CPU时间用于构建接触列表；该过程是对相邻单元进行双重循环，因此可以进行细粒度并行化。排序扫描算法只支持粗粒度并行化（各方向彼此独立）[6], 我们尝试通过划分邻域列表来对重排序进行细粒度并行化，但均未成功，不仅在 MATLAB 中如此，在 FORTRAN 中也是如此 [27]。这意味着，树代码的标量版本不仅更快，而且其标量部分更少，从而减少了根据阿姆达尔定律导致并行化效率降低的部分，也可以使用更多处理器。此外，树代码中自身支持细粒度并行化的代码占比也更高。

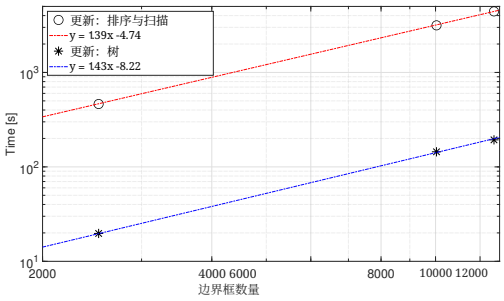


图10：树代码的更新部分（在 Xeon56 上执行）所需时间仅为排序与扫描算法的十分之一。在这些数据中，不包括包围盒的创建和更新，因此与图9相比，耗时更短。

4.2. 缓存大小对性能的影响

对于使用大量数据的程序，例如计算机模拟，限制因素并非中央处理器的时钟频率，而是内存之间的数据传输速度；后者受缓存大小、总线速度、缓存行宽度（即在内存与中央处理器之间传输数据的总线）以及数据加载

表 1：规格　　处理器的 n。多边形模拟的力计算通过线程并行化进行，而对 nei 的评估　　ghborhood 例程未进行并行化。

处理器	英特尔® 至强®		Apple M2		Apple M4	
	Xeon32 E5-2667v2	Xeon56 E5-2680v4	P-core	E-core	P-core	E-core
时钟频率 [GHz]	3.3	2.4	3.5	2.4	4.4	2.6
处理器/核心数	2 / 32	2 / 56	1 / 4	/ 4	1 / 4	/ 6
最大线程数	64	112	8		10	
内存	DDR3	DDR4	LPDDR5		LPDDR5X	
L1缓存 [KB]	512	896	128	64	128	64
L2缓存 [MB]	4	7	16 个共享，	4 个共享	16 共享，	4 共享
三级缓存 [MB]	50	70	未披露			

和存储策略的影响。仅凭处理器的原始规格很难从理论上预测性能，而且这些规格往往根本无法获取。将运行在 Xeon32 和 Xeon56 处理器上的树代码进行压缩时（见图 8），时钟频率更快的处理器反而耗时更长，这令人意外，也反映了内存技术（DDR3 与 DDR4）的差异。需要警惕的是，即便仅运行邻域算法，性能也可能更多地受到内存管理而非中央处理器时钟频率的影响。

5. 内联

调用计算机程序中的子函数时，调用例程的数据被“压入栈”（内存）会耗费一些时间，此时中央处理器只处理属于该函数的数据。当程序返回调用例程时，则必须从栈中取回其数据。内联是指将子函数直接写入调用例程，这样可以避免在栈中传入和取回数据所造成的时间损耗。其缺点是，此时必须确保调用例程和被调用子函数的所有数据都能保留在缓存中。数据量较大时，可能会导致缓存未命中和执行时间变长。换言之，这两种机制相互制衡，因此很难预先判断内联是否会提升性能。在 MATLAB 中（与许多通过设置优化标志来实现这一点的编译型语言类似），可以对代码进行内联。计算机会尽量将更多数据保留在缓存中，但当数据量超过缓存大小这一临界值时，就必须将数据换入内存。图 11 显示，对于规模约为 5000 个粒子的系统（包围盒），耗时高于拟合曲线；在这一区域，出现了一个交叉点：采用内联的树代码比不采用内联的效率更高。对于小型系统（最多几千个粒子），内联并无益处（甚至可能因其开销而产生反效果）。只有对于超过 10000 个粒子的大规模系统，内联才开始带来收益。仅 5000 个包围盒就已对应约 5000·8·4kB 的数据量。尽管这仅为缓存大小（896 KB）的 1/6，但表 2 展示了不同系统规模下的实际节点数。即使将 87381 个节点仅按 4 字节整数计算，加上包围盒所需的数据，所需内存也接近缓存大小。对于 5058 个包围盒，缓存未命中导致图 11 中采用和未采用内联的两种实现，其数据点都高于拟合曲线。

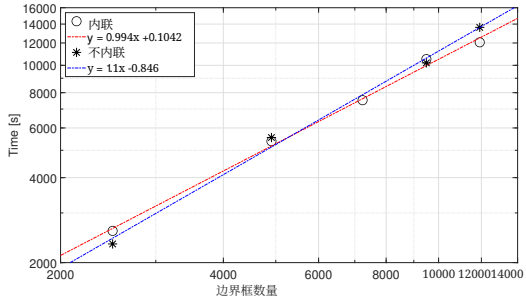


图11：树代码在内联与不内联条件下的执行时间（在Xeon56上执行）。

6. 环形复杂度比较

到目前为止，我们只讨论了计算复杂度，即随着数据维度（在我们的案例中为包围盒数量）的变化，所需工作量（以计算时间或

表 2：边界框数量、树的深度以及树中所需节点的数量。

包围盒数量	最大树深度	最大节点数
2522	9	87381
5058	9	87381
7599	10	349525
10014	10	349525
12651	10	349525

运算次数）如何变化。选择算法时另一个值得关注的问题是程序结构的复杂性。它会影响代码的实现、维护、修改和扩展，通常用“圈复杂度”[8]来衡量。与其关注原始研究中复杂的图论细节，不如聚焦于“源代码函数中的决策逻辑量”[9]。每段代码（主程序、子程序、函数）的初始值均为 1；每出现一个决策元素和一种结构（if、for、while 等），以及与之相关的逻辑运算符（参见例如 [28] ），数值就增加 1。在 MATLAB 中，可以通过 checkcode 函数自动测量圈复杂度。表 3 展示了我们的树代码在内联和未内联情况下，以及排序扫描方法的圈复杂度。由于我们的邻域算法中包含多个嵌套循环和 if 条件，其复杂度相当高。正如预期，内联会增加复杂度，因为被调用函数的所有功能都会转移到调用函数中。典型的分类对圈复杂度的评估大致如下 [29]:

1-4：低复杂度 5-7：中等复杂度。可以接受，但并非理想。8-10：高复杂度。应重构以便于测试。11-49：极高复杂度。测试和维护都非常困难。重新设计和/或重写。≤ 50：“无法测试”[28]。

这一分类与编程入门课程的教学原则相一致，这些原则强调简单性而非性能。这种方法在一般软件工程中是有益的，例如应用开发：在这类场景中，处理多样数据类型的能力至关重要，但数据集相对较小，依赖项也较少。相比之下，在科学与工程编程中，情况则明显不同：数据类型的范围通常较窄，但所处理的数据量要大得多。此外，应对科学与工程挑战的计算机模拟往往需要在较长时间内处理大量数据集，从数天到数周甚至数月不等。鉴于这些差异，在分析离散元模拟时，应谨慎应用基于圈复杂度的分类。我们最初编写的采用“低复杂度函数”的树代码程序版本始终慢于排序与扫描算法。我们的树代码最终版本在有内联和无内联情况下，以及排序与扫描方法的圈复杂度如表 3 所示。由于我们的邻域算法中存在各种嵌套循环和 if 条件，复杂度相当高。对于排序与扫描以及未内联的树代码，圈复杂度约为 70。对于带内联的树代码，复杂度显著更高，达到 273，因为被调用函数的全部功能都被转移到调用函数中。尽管一般编程教学会认为这两种方法都无法测试，但离散元模拟中的性能考量使得代码的这种复杂度成为必要。

表 3：树代码与排序-扫描算法的圈复杂度比较。

	排序与扫描	树代码	
		内联	不内联
初始化	4	19	19
更新数据结构	64	45	25
更新接触列表	2	209	33
总计	70	273	77

7. 解释执行与编译执行的 MATLAB 代码比较

为方便起见，到目前为止，我们一直使用 MATLAB 解释器代码进行性能分析。这是基于以往的经验：MATLAB 解释器代码执行线性代数运算时，并不比编译代码慢很多。这是因为 MATLAB 会预编译代码块，同时其内部函数也经过预编译，并针对基于数组的运算进行了优化。然而，在

研究过程中，我们发现 DEM代码在速度方面存在显著劣势，因为其中包含大量 if 条件和其他非算术运算。因此，我们通过消除 MATLAB 中“C 语言不友好”的结构，重构了仿真，使代码能够借助 MATLAB 的“Coder”转换为 C 代码。随后，这些C代码通过 MATLAB 的 mex命令进行编译；在本例中，为在 Apple silicon M4 上实现，使用了 Apple 的 C编译器，并启用了 -O2 优化选项。之后，目标文件被链接到 MATLAB主程序中，并通过 MATLAB图形用户界面与其余 MATLAB代码（用于绘图等）一同执行。如图 12 所示，编译代码的速度约比解释代码快一个数量级。随着系统规模增大，加速比也随之提高，这给人的印象是，编译代码比解释代码更善于避免缓存未命中。使用 -O3 进行的对照运行未带来显著变化。

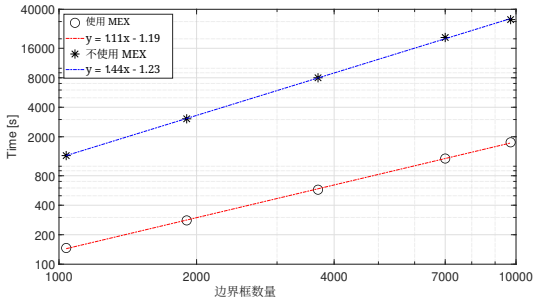


图 12：编译型（“使用 MEX”）与解释型（“不使用 MEX”）仿真的执行时间对比。加速比从最小系统的约 8 倍变化到最大系统的约 18 倍。基准测试在 MATLAB R2025b 中使用 Apple M4处理器执行，该处理器应能为解释器代码提供最有利的速度。

英特尔与Apple silicon芯片的性能数据见图13。M2的性能提升与其P核心较高的时钟频率成比例，而M4相对于时钟频率的性能接近快一倍，但在较大系统中性能略有下降。这表明其内部架构更为复杂，因为内存带宽（M2最高为100 MB/s，M4为120 MB/s）相当。

8. 结论

我们已经表明，离散元模拟的邻域搜索不仅可以借助排序扫描完成，也可以借助“最小”树代码在线性时间内完成，表明其复杂度为O(N)–。对于由

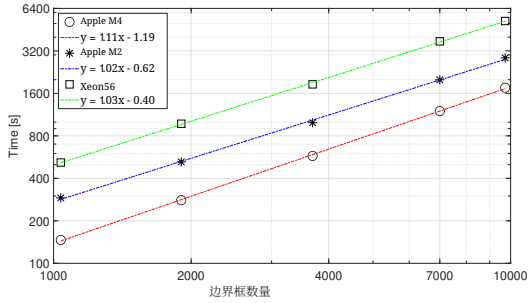


图13：使用 MEX文件（编译执行）时，Xeon56、M2 和 M4芯片上树代码的执行时间。M2 的速度约为 Xeon56 的 1.8 倍，与 M2 的 P核心时钟频率比（约 1.8）接近。对于小型系统，M4 的速度约为 Xeon56 的 3.5 倍；对于大型系统，这一倍数降至 3 倍。

对于大量移动粒子，树代码在二维中具有速度优势，同时也更有利于并行化。对于尺寸分散度较大的系统，必须将较大的粒子分解为较小的包围盒，然后在树中分别处理。对于所使用的两个模型，不同系统规模和算法的相对性能并未发生变化，因此我们认为，对于更新的内存类型和处理器，这些趋势同样适用。尽管我们对系统几何结构进行了调整，使其有利于树代码，但在三维中，这一优势可能更加明显，因为在线性维度上，不相关包围盒几何结构的数量会增加。对于由近乎静止的粒子组成的大型构型，树代码的优势会变得不那么显著。对于运动剧烈的离散元系统（例如颗粒气体），性能提升应当十分明显，因为大量邻域记录工作只对应相对少量的相互作用计算。对于类似光滑粒子流体动力学和移动粒子半隐式法中那样会相互穿透的粒子，由于邻域变化很大，因此既不建议使用树代码，也不建议使用排序扫描算法。另一个可能的应用领域是有限元法中的自适应网格划分算法，因为其邻域相对有限。除用于结构力学之外 [30], , 树中包含的几何信息还可用于帮助构建具有大幅变化流动的流体动力学网格，或构建粒子之间的网格。然而，用于邻域计算的树代码需要付出显著的算法复杂性代价；根据计算机科学的评价标准，其复杂性甚至超出了“无法测试”的程度。

利益冲突声明

作者声明，他们不存在任何已知的可能影响本文所报告工作的利益冲突或个人关系。

代码可用性

四叉树的示例实现可在 https://github.com/DKrengel/ Matlab_Quadtree 获取。该示例模拟了旋转滚筒中 ~ 1600 自由多边形颗粒的运动。为便于可视化，该程序使用 MATLAB 实现。

致谢

D. Krengel 和 J. Chen 谨此感谢日本学术振兴会（JSPS）通过科研费助成事业（JP21H01422、JP21KK0071 和 JP21K04265）提供资助。

参考文献

[1] H.-G. Matuttis, J. Chen, 《理解离散元法：颗粒及多体系统中非球形粒子的仿真》，第1版，Wiley，新加坡，2014。[2] B. C. Vemuri, Y. Cao, L. Chen, 《快速碰撞检测算法及其在颗粒流中的应用》，《Computer Graphics Forum》17（2）（1998）121–134。doi:10.1111/1467-8659.00233。[3]B. Vemuri, L. Chen, L. Vu-Quoc, X. Zhang, O. Walton, 《颗粒流模拟中高效且准确的碰撞检测》，《Graphical Models and Image Processing》60（6）（1998）403–422。doi:10.1006/gmip.1998.0479。[4] P. A. Cundall, O. D. L. Strack, 《颗粒集合体的离散数值模型》，《Géotechnique》29（1）（1979）47–65。[5] J. Chen, D. Krengel, D. Nishiura, M. Furuichi, H.-G. Matuttis, 《基于JKR理论的力–位移关系：用于黏附颗粒的离散元模拟》，《Powder Technology》427（2023）118742。doi:10.1016/j.powtec.2023.118742。

[6]J. Chen, H.-G. Matuttis, 《多核计算机上用于凸多面体的离散元法代码优化与 OpenMP 并行化》，《国际现代物理学杂志 C》24 (02) (2013) 1350001。doi:https://doi.org/10.1142/S0129183113500010。

[7] J.-F. Hake, W. Homberg, 《内存组织对矩阵计算性能的影响》，《并行计算》17 (2) (1991) 311–327。
d
oi:
https://doi.org/10.1016/S0167-8191(05)80116-4。

[8]T. J. McCabe, 《一种复杂度度量》，《IEEE 软件工程汇刊》SE-2 (4) (1976) 308–320。doi:http://dx.doi.org/10.1109/TSE.1976.233837。

[9] A. H. Watson, T. J. McCabe, 《结构化测试：一种使用圈复杂度度量的测试方法》，技术报告 (1996)。[10]J. J. Monaghan, 《光滑粒子流体动力学》，《天文学与天体物理学年度评论》30 (1992) 543–574。doi:10.1146/annurev.aa.30.090192.002551。

[11]S. Koshizuka, Y. Oka, 《用于不可压缩流体碎裂的移动粒子半隐式方法》，《核科学与工程》123 (1996) 421–434。doi:https://doi.org/10.13182/NSE96-A24205。[12] D. Krengel, J. Chen, Z. Yu, H.-G. Matuttis, T. Matsushima, 《为多面体离散元颗粒实现范德华力》，《IOP地球与环境科学会议系列》1480 (1) (2025) 012030。doi:10.1088/1755-1315/1480/1/012030。URL
https://doi.org/10.1088/1755-1315/1480/1/012030

[13] D. Baraff, 《刚体模拟》，载于：M. Kass（编），《SIGGRAPH 93：物理建模导论》，ACM，1993，第 H1–H68 页。[14] J. D. Cohen, M. C. Lin, D. Manocha, M. Ponamgi, 《I-COLLIDE：面向大规模环境的交互式精确碰撞检测系统》，载于：《I3D ’95：1995年交互式三维图形研讨会论文集》，计算机协会，纽约，NY，美国，1995，第 189–197 页。

[15] R. A. Finkel, J. L. Bentley, 《四叉树：一种用于复合键检索的数据结构》，《Acta Informatica》4 (1) (1974) 1–9。doi:10.1007/bf00288933。

[16]J. L. Bentley，《用于关联搜索的多维二叉搜索树》，ACM通讯 18 (1975) 509–517。doi:10.1145/361002.361007。

[17]A. W. Appel，多体仿真的高效程序，SIAM科学与统计计算期刊 6 (1) (1985) 85–103。doi:10.1137/0906008。

[18] J. Barnes、P. Hut，一种分层的 o(n log n) 力计算算法，自然 324 (6096) (1986) 446–449。doi:10.1038/324446a0。[19] M. C. Lin，用于动画和机器人学的高效碰撞检测，博士论文，加利福尼亚大学伯克利分校，伯克利，加利福尼亚州 (1993)。[20] M. Wackenhut，模拟极多分散介质的快速算法，博士论文，斯图加特大学数学与物理系 (2006)。[21] S. R. Schwartz、D. C. Richardson、P. Michel，将软球离散元方法实现于高性能并行引力树编码中，颗粒物质 14 (3) (2012) 363–380。doi:10.1007/s10035-012-0346-z。

[22] J. G. Stadel，宇宙学N体仿真及其分析，博士论文，华盛顿大学（2001）。

[23] D. Richardson，星子动力学的直接大规模N体仿真，《伊卡洛斯》143 (1) (2000) 45–59. doi:10.1006/icar.1999.6243.[24] J. Duriez、C. Galusinsky，任意颗粒形状颗粒材料的八叉树水平集表示，载于：《流体力学专题问题2020》，TPFM，热力学研究所，捷克科学院，v.v.i.，2020，第64–71页。

[25] D. Krijgsman、V. Ogarko、S. Luding，任意多分散粒子系统中用于接触检测的分层网格数据结构的最优参数，《计算颗粒力学》1 (3) (2014) 357–372. doi:10.1007/s40571-014-0020-9.

[26]D. Krengel、H.-G. Matuttis，《二维多体问题中静摩擦的实现》，《日本物理学会志》87 (12) (2018) 124402。doi:https://doi.org/10.7566/JPSJ.87.124402。

[27] R. Tenhagen、H.-G. Matuttis，球形粒子系统的并行化，技术报告（2013）。

[28]TheMathworks，使用圈复杂度度量代码复杂度，https://www.mathworks.com/help/matlab/matlab_prog/measure-code-complexity-using-cyclomatic-complexity.html，最后访问：2024年2月（更新：2023年10月23日）。

[29]Incusdata，圈复杂度，https: // incusdata.com/blog/cyclomatic- complexity，最后访问：2024年2月（修订：2023年7月19日）。

[30]C. Lee、S. Natarajan，面向准不可压缩超弹性固体的自适应四叉树多边形基边光滑有限元法，《边界元工程分析》155（2023年）973–994。doi:https://doi.org/10.1016/j.enganabound.2023.07.003