ACM中国一国际并行计算挑战赛

ACM-China International Parallel Computing Challenge

目录 CONTENTS

1. 参赛队伍简介
2. 应用程序运行的硬件环境和软件环境
3. 应用程序的代码结构
4. 优化方法
5. 程序运行结果
6. 其他可行的优化方法

01 参赛队简介

- 队伍名称：网速好快
- 队伍编号：IPCC20224013
- 参赛单位：中国科学院 计算机网络信息中心
- 参赛队员：邱霁岩、陈宇轩、白晨晗、韩子栋(排名不分先后)
- 指导老师：张鉴

02 应用程序运行的硬件环境和软件环境

- 硬件设备
- CPU：AMD EPYC 7452 32-Core Processor
- 双节点，每节点双socket，每socket 32核心

- 软件环境：
- OS：CentOS Linux release 7.9.2009
- GCC compiler：GCC-8.1.0

	原始程序	优化后
Case 1	492.504 s	0.081s
Case 2	约9小时	0.104s
加速比 Case 1	1.000	6080.296
加速比 Case 2	1.000	311538.462

03 应用程序的代码结构

假设要处理的数据集是$S = \{x_i \mid i = 1, 2, \dots, n\}$，共有$n$个数据点，任两点间的距离可以由距离函数$d(.,.)$计算；要选择$k$个点作为支撑点，标记为$P = \{p_j \mid j = 1, 2, \dots, k\}$。对于$S$中任意的一个数据点$x$，其基于支撑点集合重建的坐标是其到各支撑点的距离形成的向量，

$$x^p = \big(d(x, p_1), \dots, d(x, p_k)\big)$$

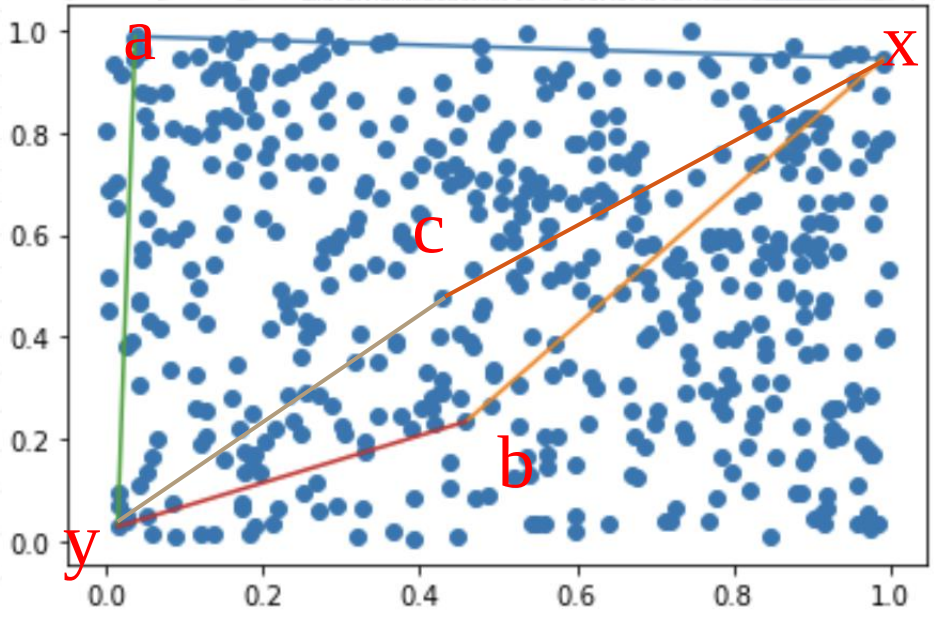
本项目采用距离和目标函数，即任意两点的重建坐标间的切比雪夫距离的和，越大越好：

$$\sum L_\infty(x^p, y^p), x, y \in S$$

$$\text{其中} L_\infty\big((a_1, a_2, \dots, a_k), (b_1, b_2, \dots, b_k)\big) = \max_i(|a_i - b_i|)$$

需要准确地求出（可采用但不限于穷举法，但要保证结果的正确性）目标函数值最大和最小的各 1000 个支撑点集合。

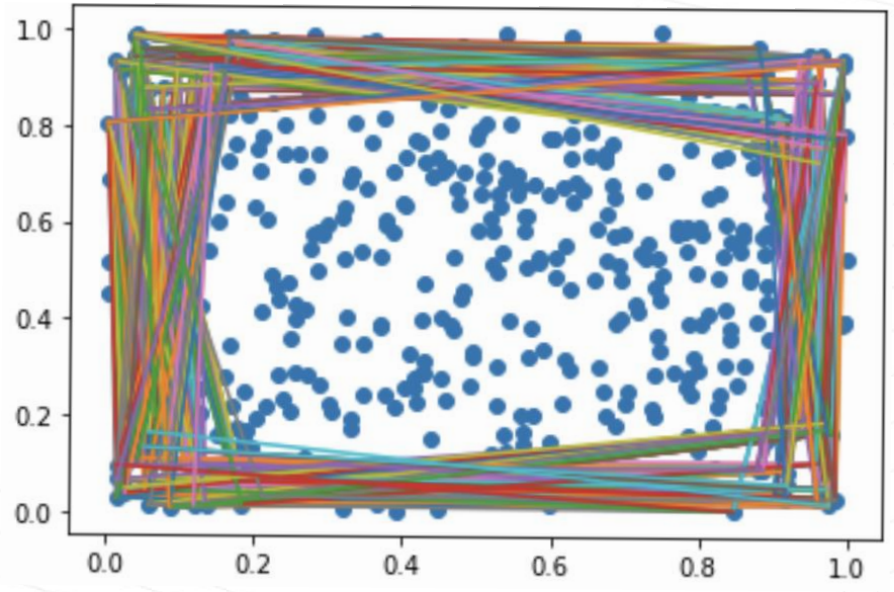
03 应用程序的代码结构



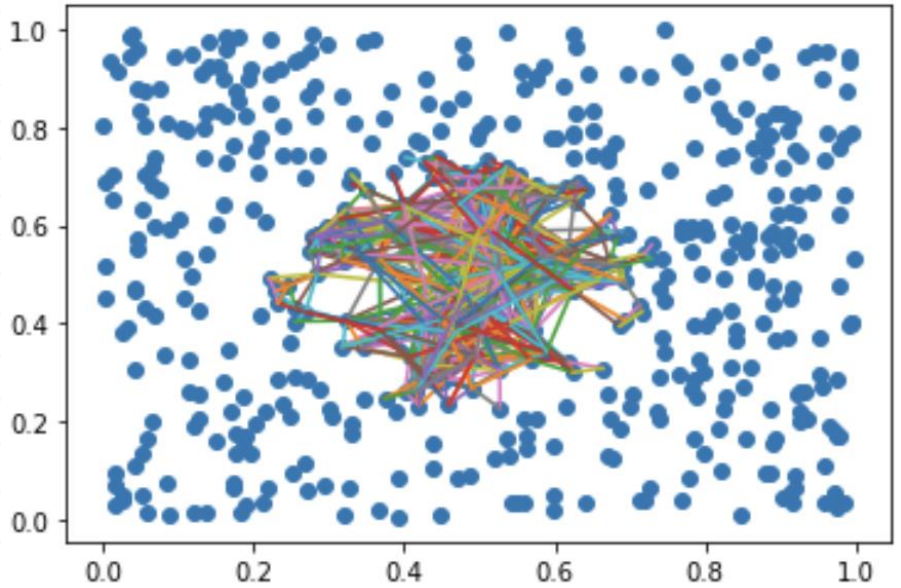
- 距离计算函数，以k=3为例

```
def L(a, b, c):
    sum=0
    for x in P:
        for y in P:
            sum = sum+ max(abs(Lax - Lay), abs(Lbx - Lby),
                           abs(Lcx - Lcy), ···)
    return sum
```

03 应用程序的代码结构



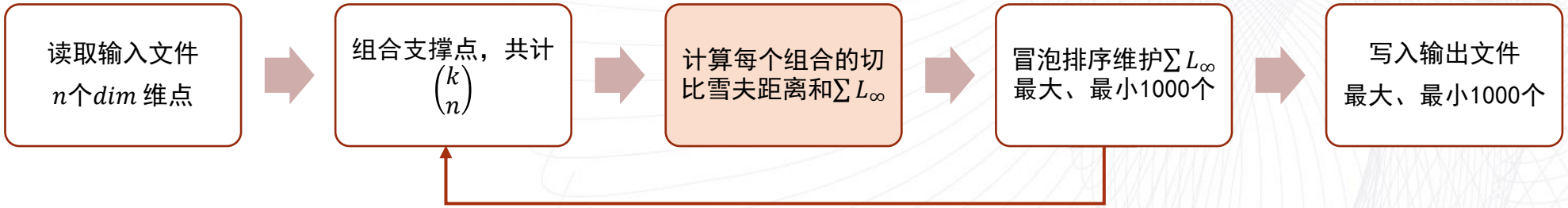
最大1000个支撑点对



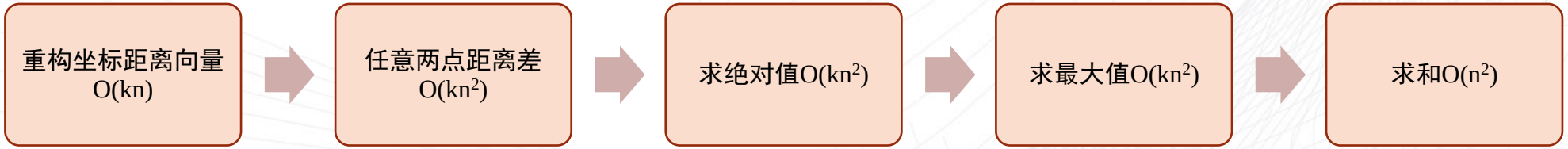
最小1000个支撑点对

03 应用程序的代码结构

- 原主程序



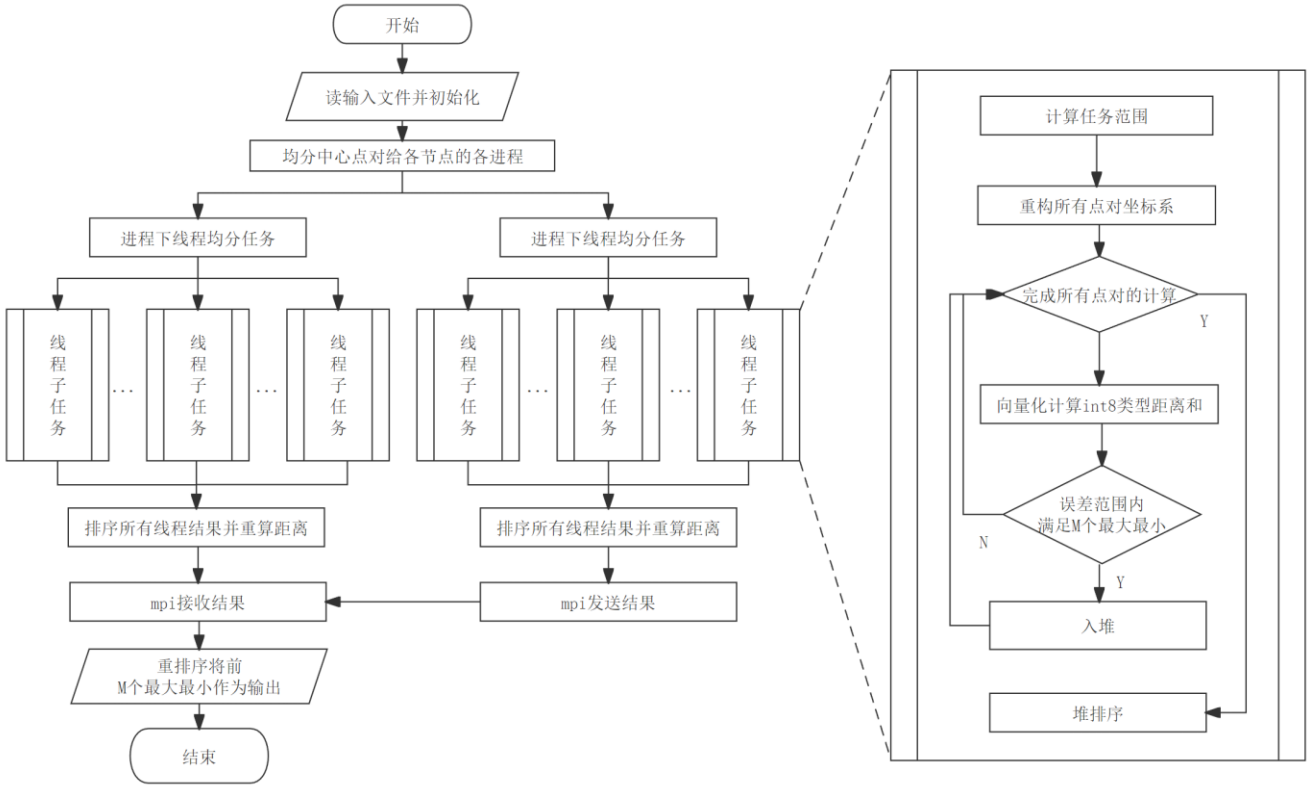
- 原程序切比雪夫距离和(时间占比98%)



03 应用程序的代码结构

应用程序的算法流程如图所示。

1. 读取文件
2. 每一个线程分配任务，开始并行计算
3. 计算完成之后，每个节点的指派一个线程合并该节点的结果
4. 使用MPI通信到0号节点再次合并结果。就可以得到最终结果。



04 优化方法

1. 编译优化
2. 算法和数据结构优化
3. 数值量化
4. 向量化
5. 并行优化

4.1 编译优化

- 默认

```
gcc –O0 pivot.c -lm -o pivot (gcc 4.8.5)
```

- 替换后

```
mpicc -std=c11 -Ofast  -march=native -o pivot pivot.c -lm
```

- 典型的编译优化：
  - 常量传播
  - 内联函数
  - 循环展开
  - …

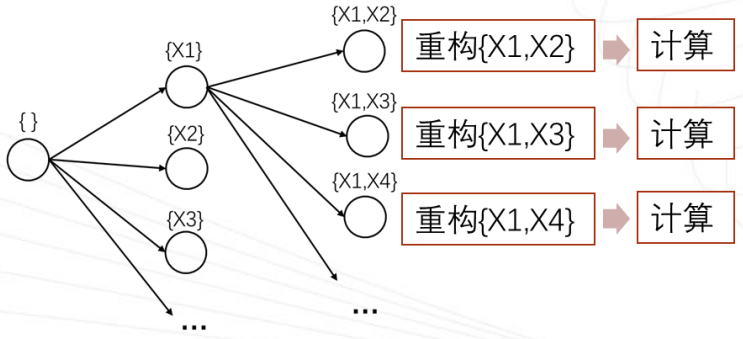
算例	优化前	优化后	加速比
CASE 1	492.504 s	48.747 s	10.103
CASE 2	约9小时	约40分钟	

4.2 算法和数据结构优化

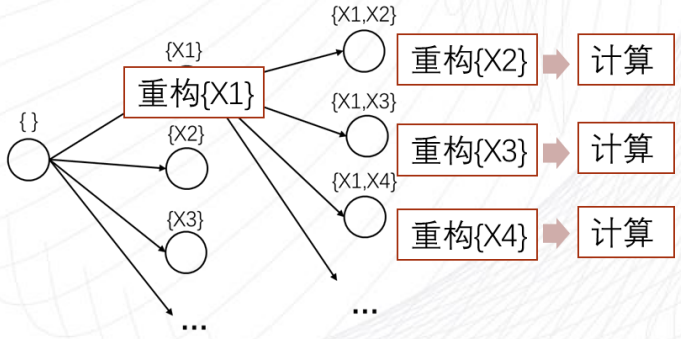
1. 根据对称性，只需要计算其中一半。

$$\sum L_\infty(x^p, y^p), x, y \in S = 2\sum L_\infty(x^p, y^p), x, y \in S, x < y$$

2. 复用坐标重构结果



原算法回溯搜索树重构过程

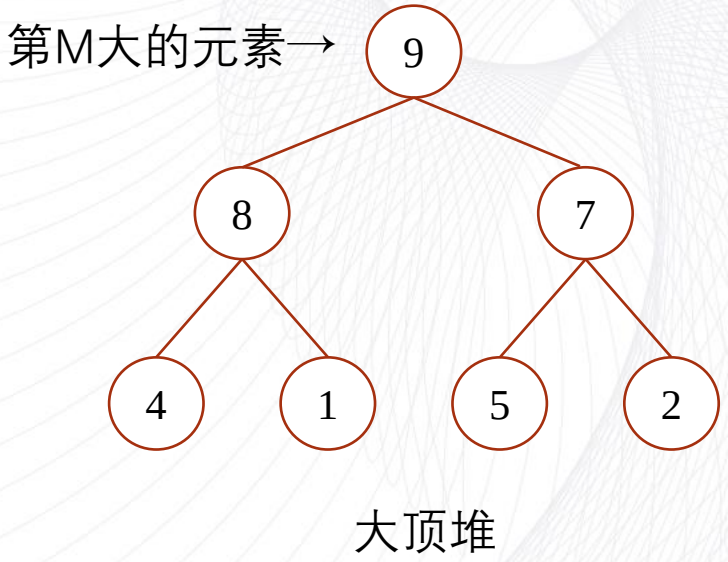


改进算法回溯搜索树重构过程

4.2 算法和数据结构优化

3. 用堆排序、快速排序代替冒泡排序

- 维护最小的M个使用大顶堆
- 维护最大的M个使用小顶堆



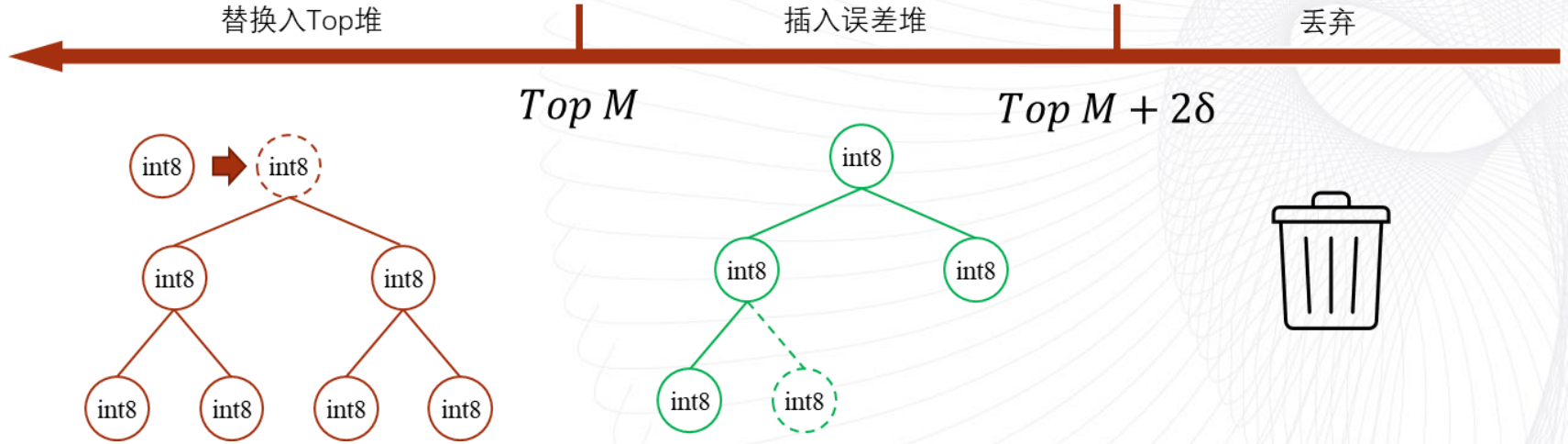
大顶堆

算例	优化前	优化后	加速比
CASE 1	48.747 s	19.659 s	2.479
CASE 2	约40分钟	大于10分钟	

4.3 数值量化 (1/2)

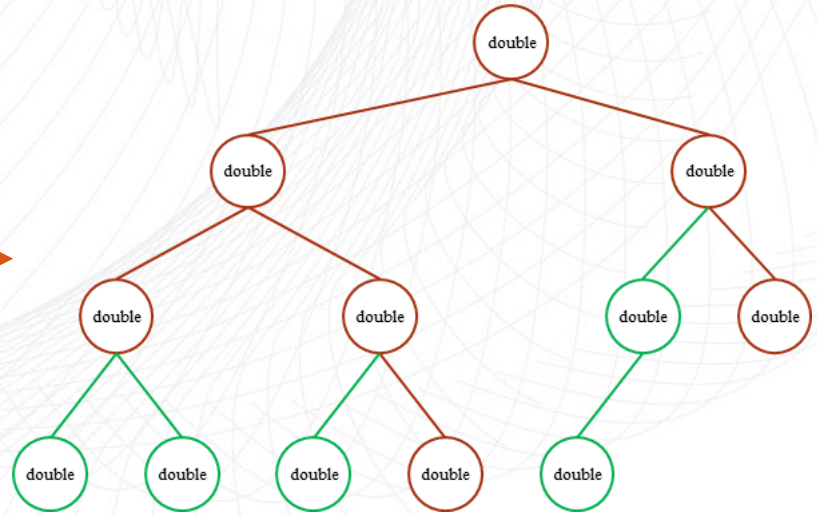
- 将范围在 [0, 1] 的实数坐标量化为范围在 [0, Q] 的整数坐标
- 对于实数 $x \in [0,1]$，量化后的结果为 $\tilde{x} = \left\lfloor Qx + \frac{1}{2} \right\rfloor$
- 利用 $\sum L_\infty(\tilde{x}^p, \tilde{y}^p)\ /Q$ 的结果估算 $\sum L_\infty(x^p, y^p)$ 的值
- 误差 $\delta = \left|\sum L_\infty(x^p, y^p) - \sum L_\infty(\tilde{x}^p, \tilde{y}^p)\ /Q\right| \leq \sqrt{dim}\binom{n}{2}/Q$
  - 求 $\sum L_\infty(\tilde{x}^p, \tilde{y}^p)\ /Q$ 最大(最小)的 M 组支撑点
  - 以及与第 M 大(小)的结果差距不超过 2δ 的支撑点
- 最后复原为实数坐标计算 $\sum L_\infty(x^p, y^p)$，排序求得最终结果

4.3 数值量化 (1/2)



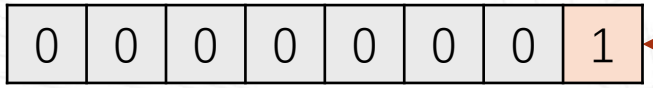
$\sum L_\infty(\tilde{x}^p, \tilde{y}^p)/Q$ 最大(小)的 M 组支撑点　　第 M 大(小)差距≤ 2δ 支撑点

合并+double重新计算

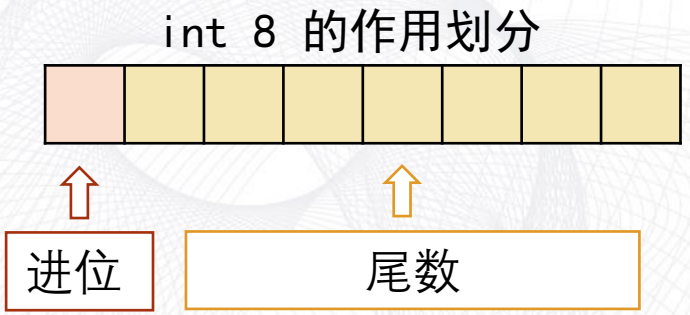


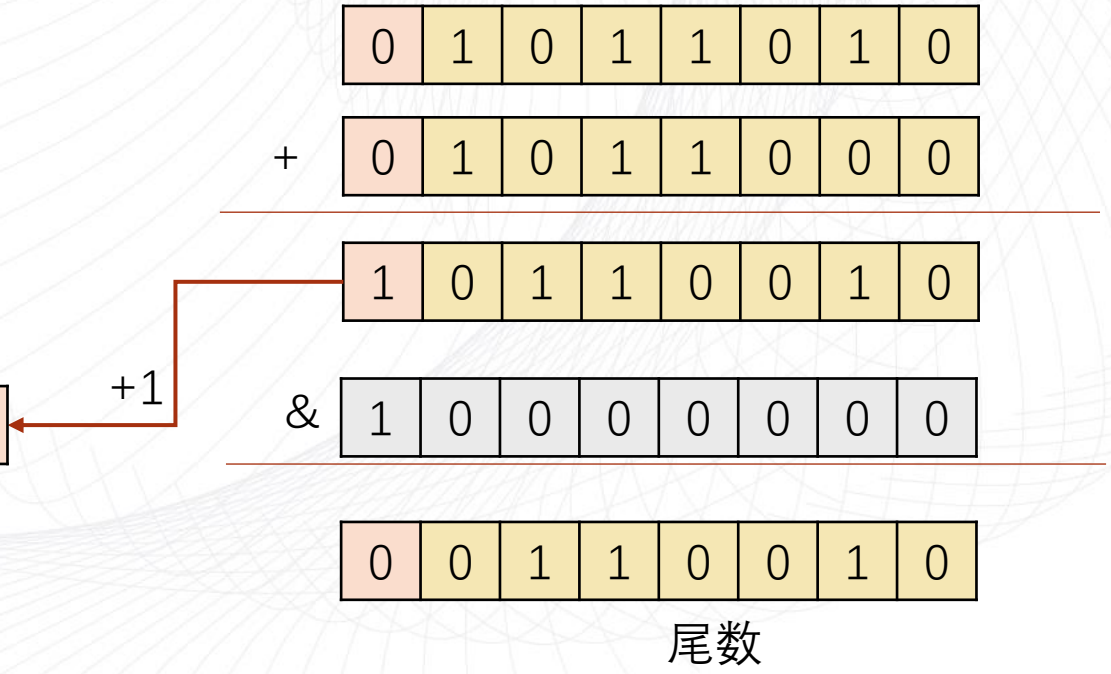
4.4 向量化(1/2)

- int8 部分：
  - 所有int8数据按照256位对齐。
  - 所有运算都转化为epi8操作。
  - 每个int8向量包含32个点。
- 累加溢出怎么办？
  - 用两个int8模拟int16
  - 使用位运算实现指令集未实现全加器。
  - 距离求和=进位次数×128+尾数。



进位次数

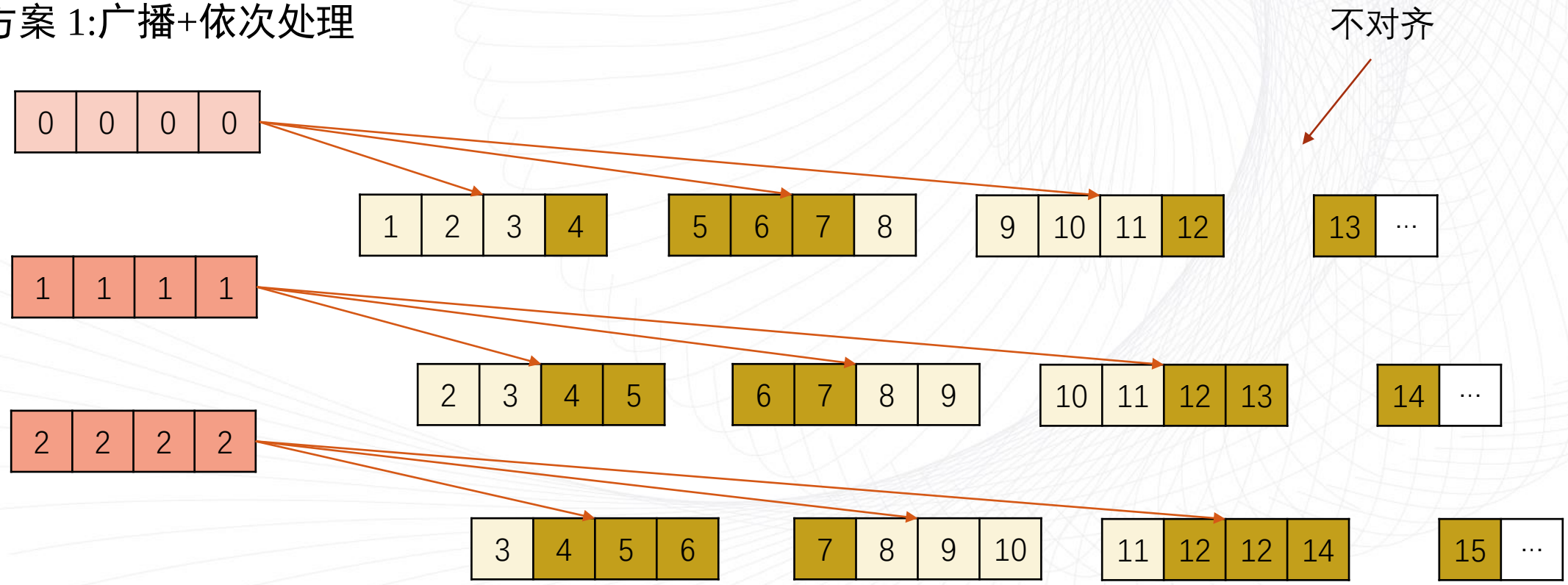




尾数

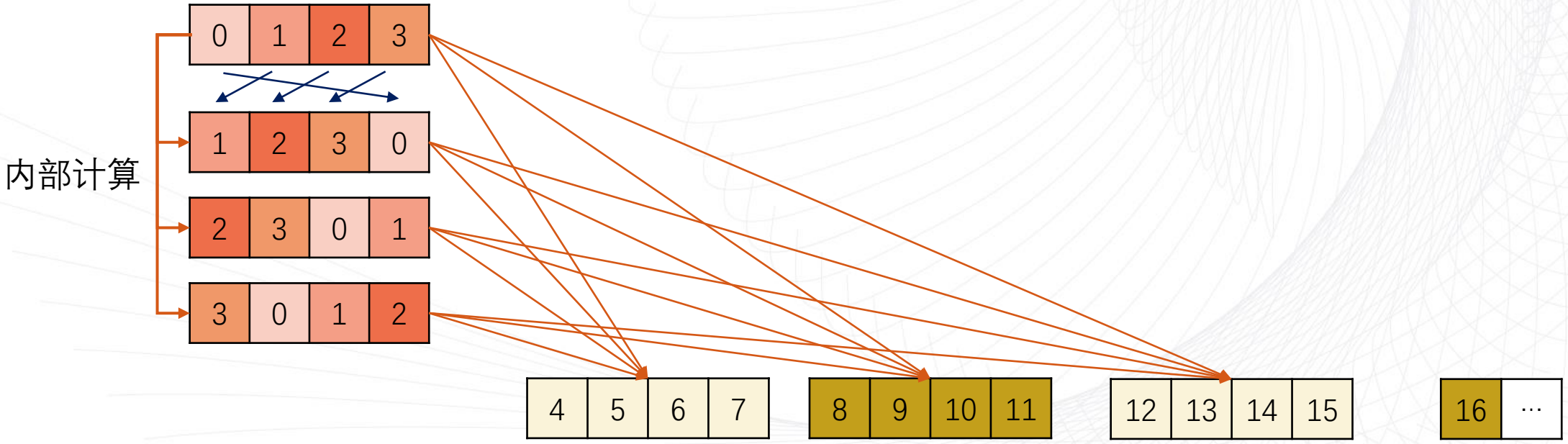
4.4 向量化(2/2)

- double 部分的瓶颈是访存
- 设计依照减少访存次数设计向量算法
- 方案 1:广播+依次处理



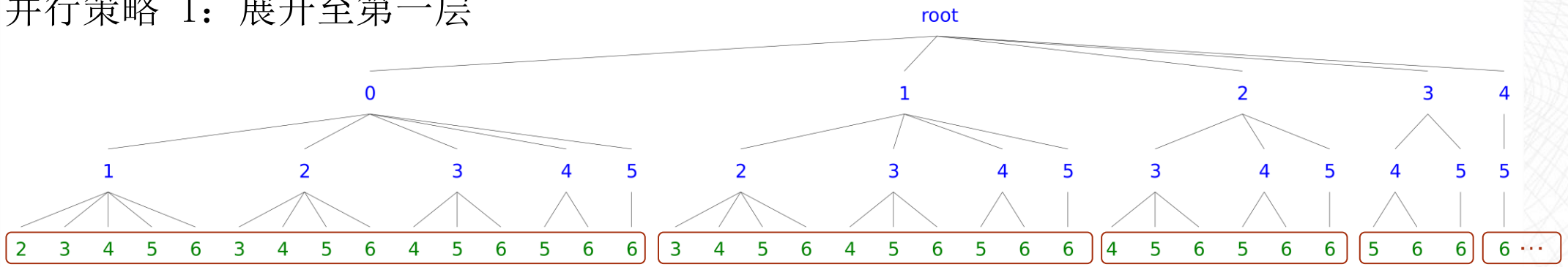
4.4 向量化(2/2)

- 方案2： 读入一次+交换位置
- 对齐、减少读入次数
- 需要处理内部计算

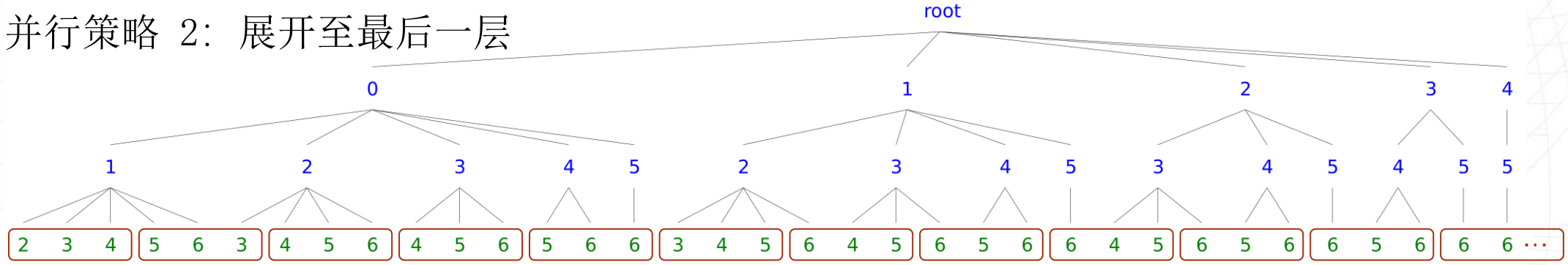


4.5 使用OpenMP并行

并行策略 1：展开至第一层



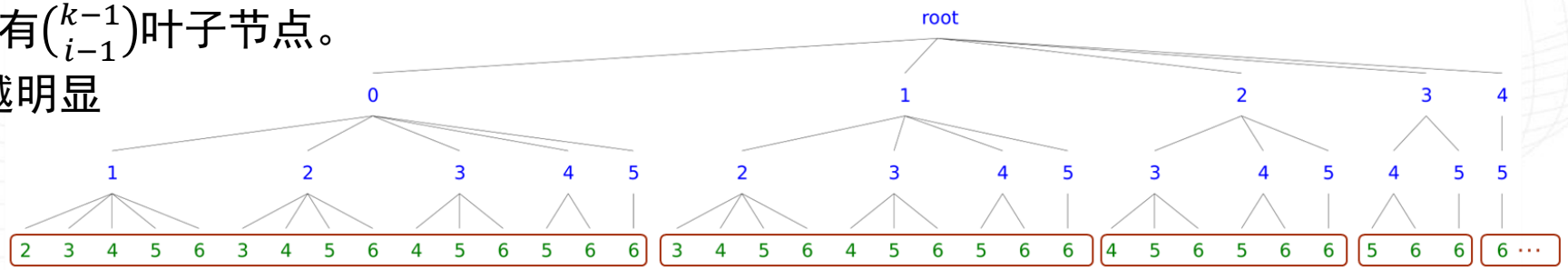
并行策略 2：展开至最后一层



4.5 使用OpenMP并行

- 并行策略 1
  1. 将递归的第一层循环并行化
  2. 为每个线程分配搜索树的第一层
  3. 相邻根节点分配在同一个线程上，尽可能多的复用父节点结果。
  4. 每个线程维护M个有序最大、最小支撑点。最后使用单线归并排序得到结果。

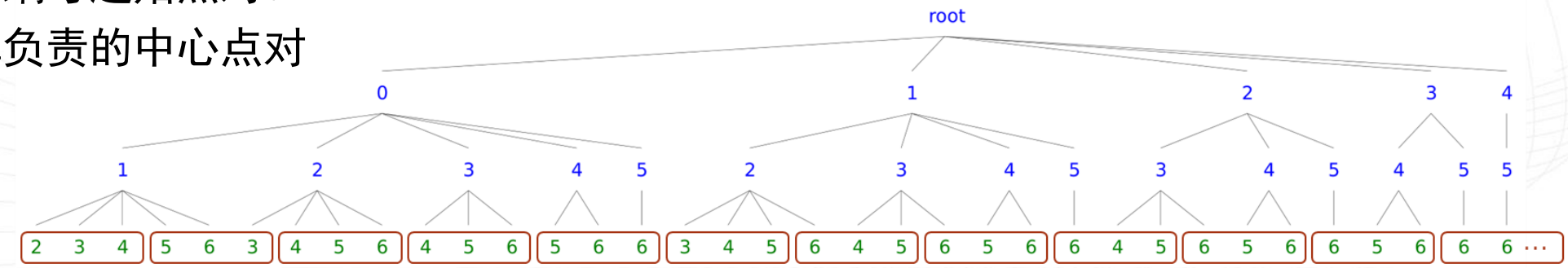
- 任务分配不均
  - 搜索树的每个子节点都大于父节点，树结构极不对称。
  - 每一层根节点个数呈现$O(n^{k-1})$级别递减。
  - 第一层有n个节点。
  - 第一层的第i个节点有$\binom{k-1}{i-1}$叶子节点。
  - K越大，不均显现越明显



4.5 使用OpenMP并行

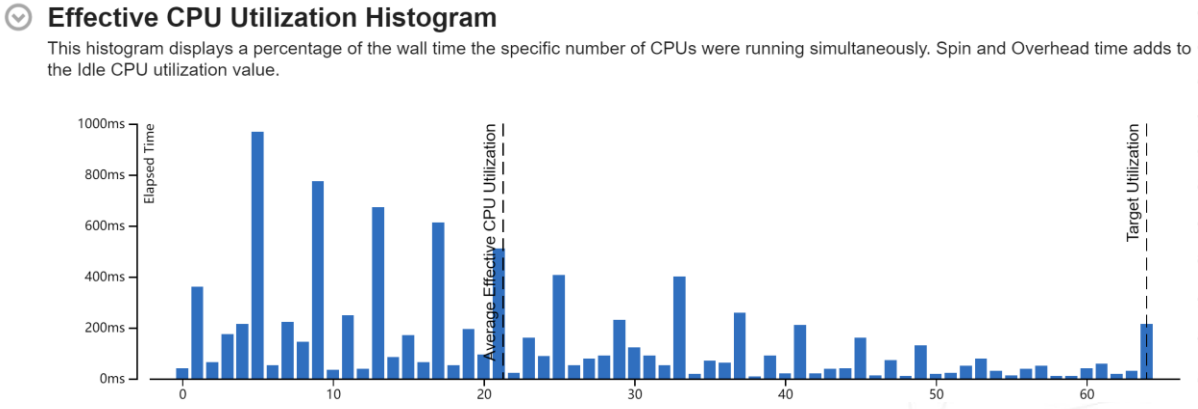
- 并行策略 2
  1. 将递归展开为循环。
  2. 为每个线程均匀分配相同多的叶子结点。
  3. 通过进位的方式递推下一个要计算的叶子结点。

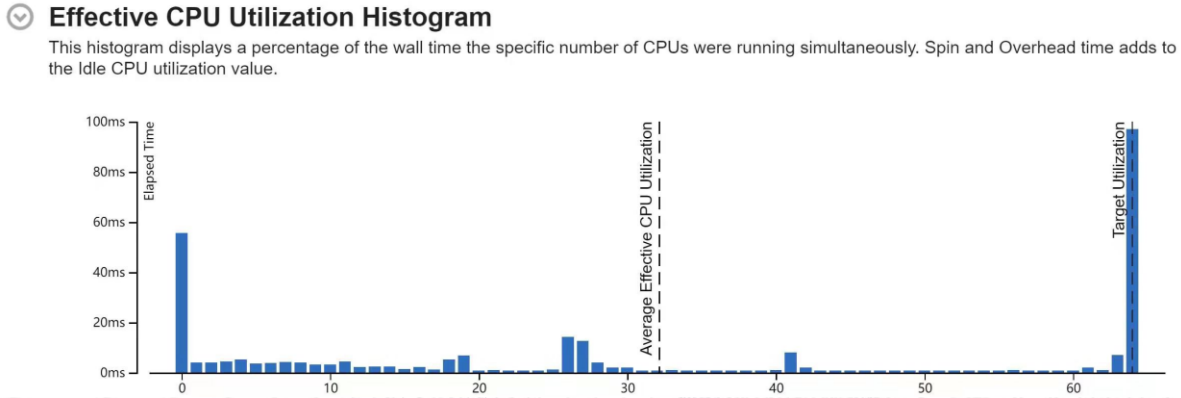
- 任务均分算法：
  - 为每个线程分配任务编号。
  - 每个任务编号起始点对。
  - 依次计算负责的中心点对



4.5 使用OpenMP并行

- 并行策略1，2效果对比





CPU效率统计图对比，策略1(左图)，策略2(右图)

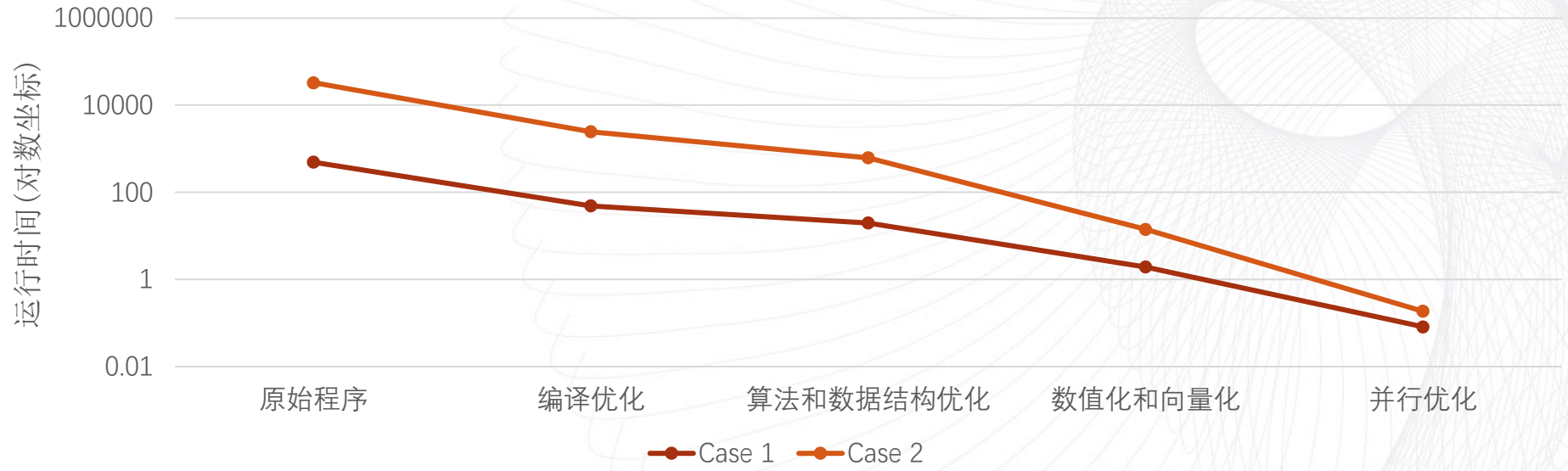
算例	优化前	策略1	策略1加速比	策略2	策略2加速比	策略2/策略1
CASE 1	1.921s	0.255 s	10.234	0.134	14.230	1.903
CASE 2	12.133 s	11.964 s	-	3.487	-	3.431

4.4 并行优化(1/2)

- MPI并行
  - 从节点向主节点发送运算结果
  - 将两节点计算结果合并。
  - 使用快速排序计算最终结果。

算例	优化前	优化后	加速比
CASE 1	1.921s	0.081 s	23.716
CASE 2	12.133 s	0.104 s	116.66

05 程序运行结果



	原始程序	编译优化	算法和数据结构优化	数值量化和向量化	并行优化
Case 1	86.611	48.747	19.659	1.921	0.081
Case 2	~32400.000	~2400.000	~600.000	12.133	0.104
加速比 Case 1	1.000	10.103	25.052	256.379	6080.296
加速比 Case 2	~1.000	~13.500	~54.000	~2670.403	~311538.462

* Case 2 包含主办方提供的估算数值

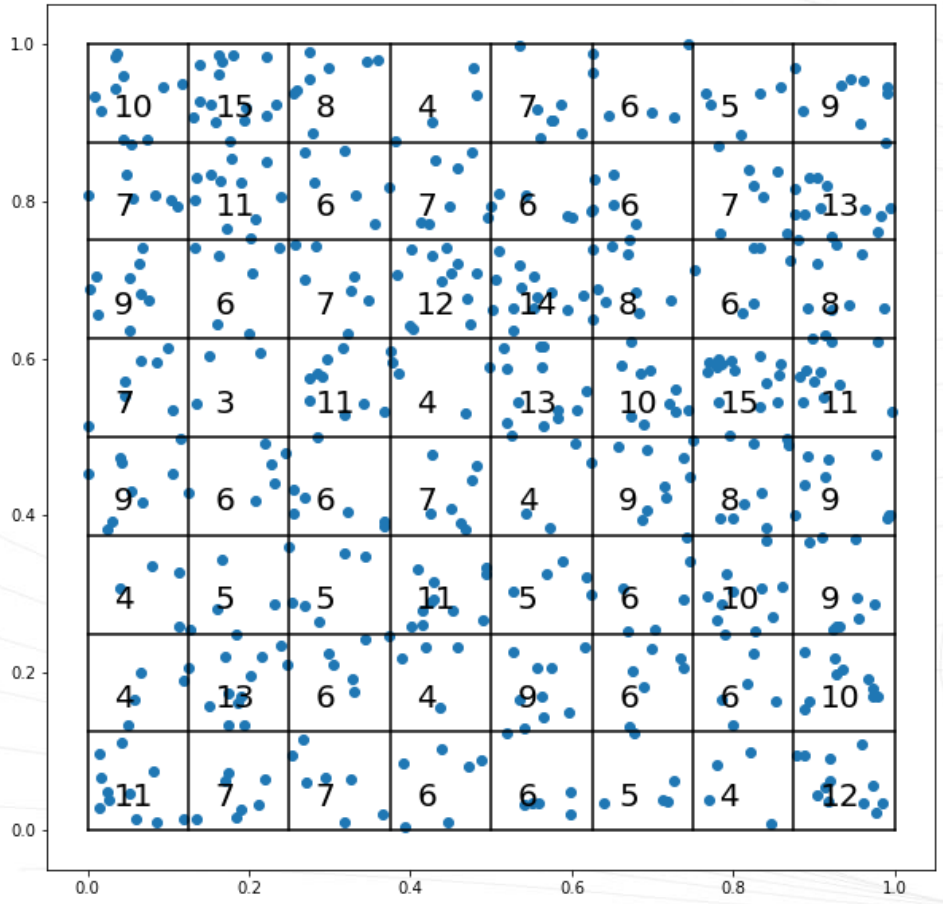
感谢观看

THANKS FOR WATCHING

06 其他优化方法

1. 网格剪枝 ——适用于n较大、dim较小情况
2. TSP剪枝——适用于点有明显聚类的情况
3. 采样计算——适用于n非常大的情况
4. SVD——适用于n较小，k较大的情况

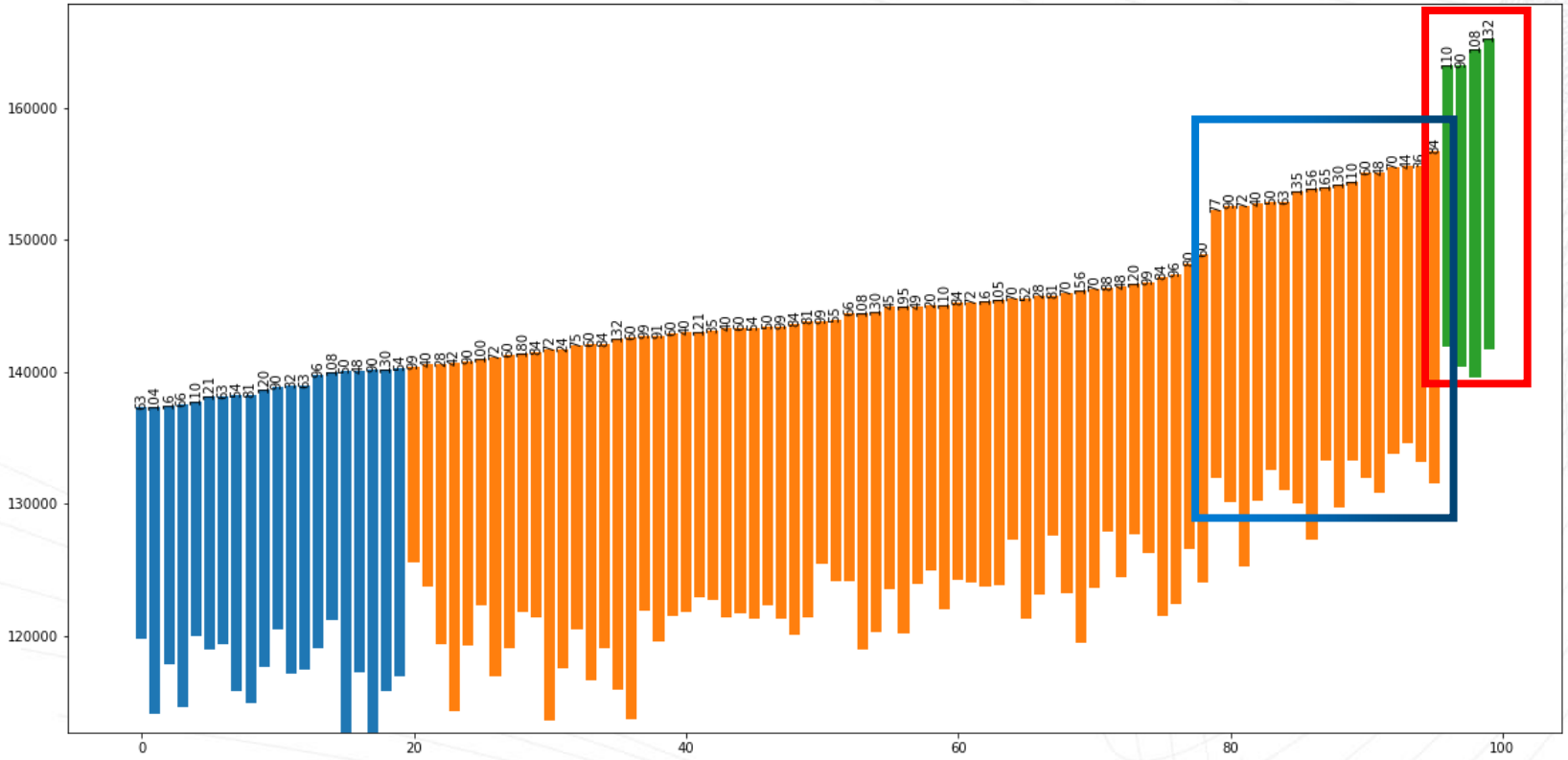
6.1 网格剪枝

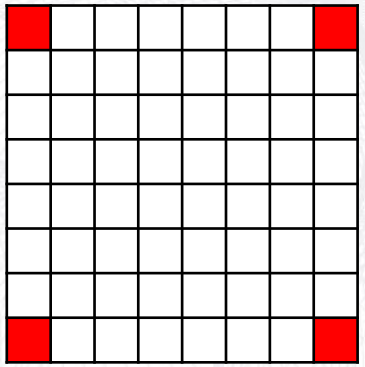


8*8网格

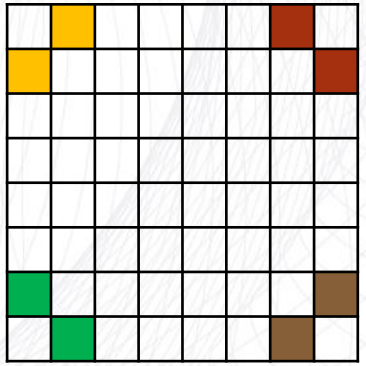
- 将所有的点划分到每个格子里，
- 选取k个格子(可重复)
- 每个格子选取k个点(不重复)
- 计算这k个格子中的k个点得到的SumDistance范围

6.1 网格剪枝





440个点对

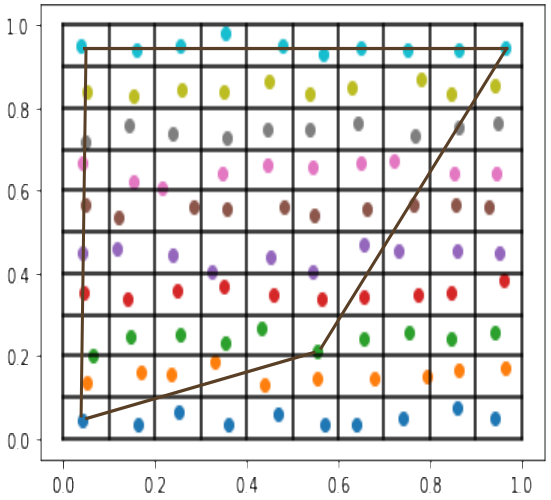


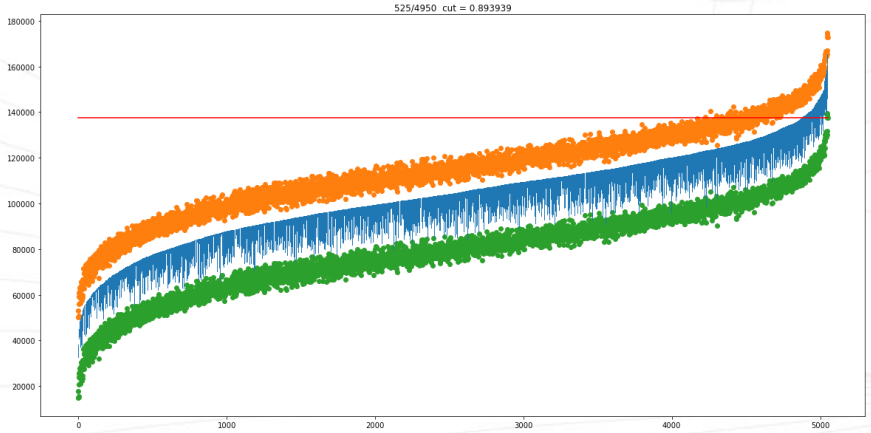
1430个点对

64个格子去组合共计2080种情况，此处按照上界排序

可以对前M个最大最小点对的SumDistance做预估

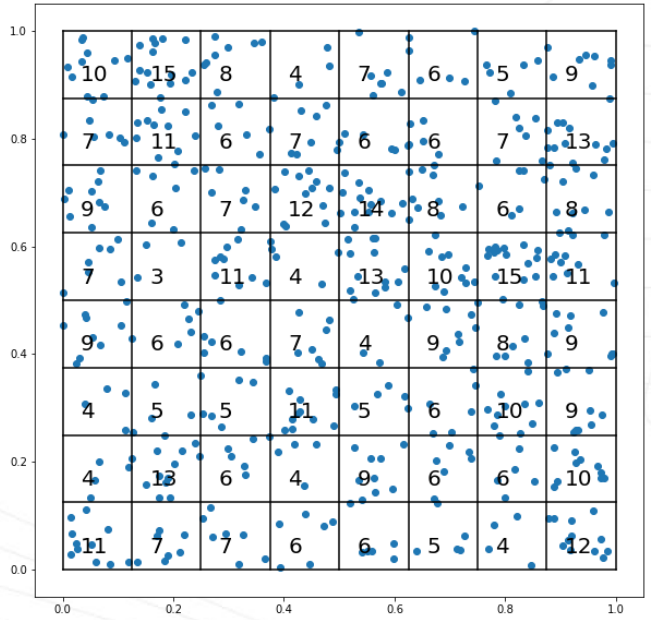
6.1 网格剪枝



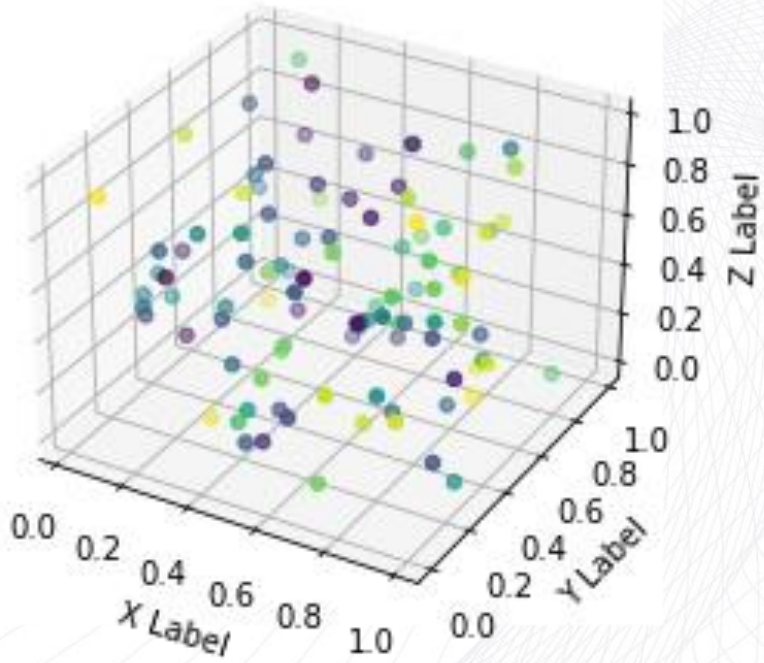


- 通过中心点的距离预估:gridSumDistance
- 是一个无偏估计
- 计算距离之后，要使用频数加权，乘以桶中点个数
- 上下界：vor=±2*格子对角线*$\binom{k}{n}$
- 时间复杂度$O(m^{k+2})$,m为格子数，远远小于总体时间复杂度$O(n^{k+2})$

6.1 网格剪枝



CASE1 最多可减少50%计算



CASE 2 点过于稀疏，效果不佳