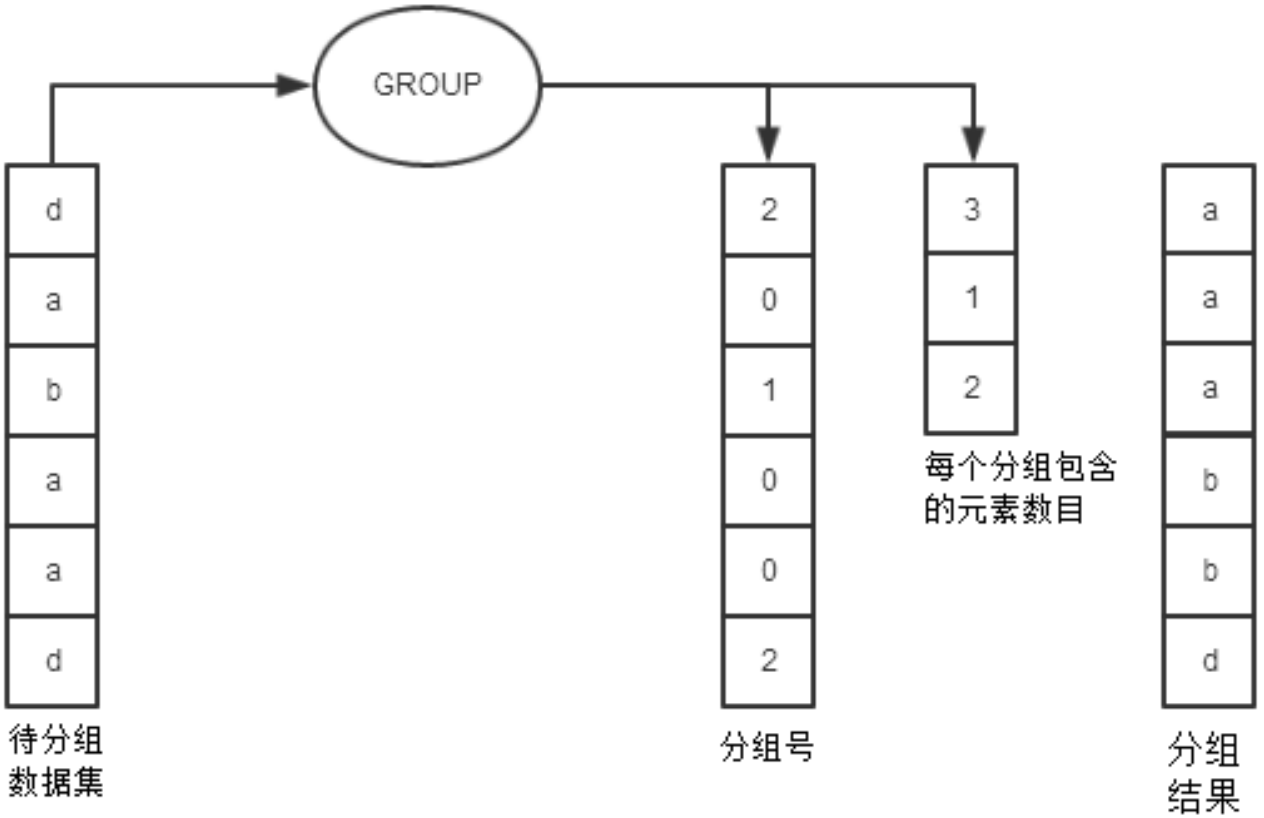
基于共享内存的数据分组算法研究

分组问题

定义：数据分组是根据统计研究的需要，将原始数据按照某种标准划分成不同的组别，分组后的的数据称为分组数据。

分组问题最常见的应用是SQL查询的Group-By操作



研究意义

- 观察数据的分布特征
  - -数据分布是否均衡
  - -数据重复率有多高

- 分组操作频繁出现在查询中，是影响查询效率的重要因素
  - -多数聚合操作与分组操作相关联。TPC-H基准测试22条查询语句都包含聚合操作，其中有16条与分组操作相关
  - -据统计，Group-By操作出现在SQL查询的频率是43.1%

研究现状

- 算法
  - 排序
  - 哈希表

- 实现方式
  - 二阶段法(Two Phrase)—分治法：划分+合并
  - 预分区法(Repartition)—划分法：数据先分区，区间无重复

主要内容

1. 基于归并排序的分组

- 改进归并过程，给出归并过程的并行化方案

2. 分段计数分组法

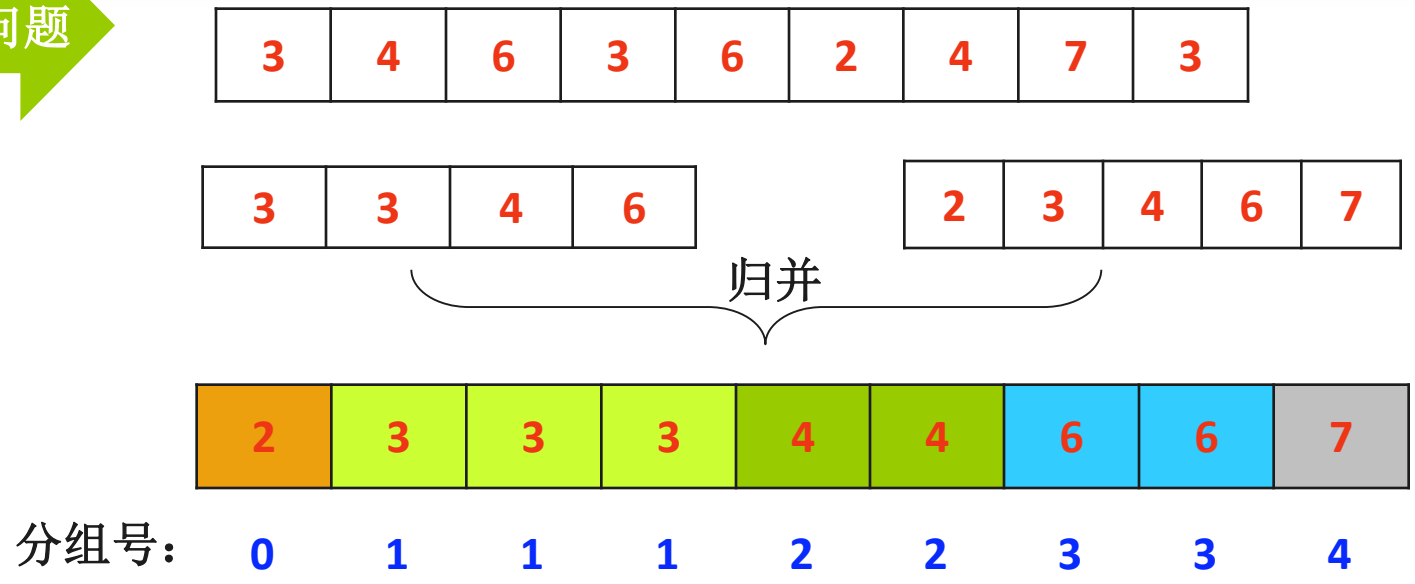
- 研究分段计数分组算法，提出分段计数分组法的并行化方案

3. 基于哈希表的分组

- 研究基于哈希表算法的分组，提出独立哈希表法的cache优化方案
- 在MIC平台上实现基于独立哈希表的分组，并进行向量化优化

基于归并排序的分组

过程及存在问题



➢存在问题

- 归并过程复杂度较大
- 以往的归并过程并行化方案效率不高

➢优化

-算法优化

-归并过程并行化

-合并归并过程和分配分组号过程

基于归并排序的分组

- 分组记录预处理，减少记录移动
  - 构造一个包含关键字的“值”和“下标”的结构体，开辟一个与源数据集长度相同的结构体数组U，将源数据集的每个元素封装成上述结构体并写入数组U中，构成分组对象
- 分组对象进行排序
  - 局部排序— QSORT
  - 归并
- 分配分组号并统计各个分组包含的记录数

关键字	d	a	b	a	a	d
下标集合(I)	0	1	2	3	4	5

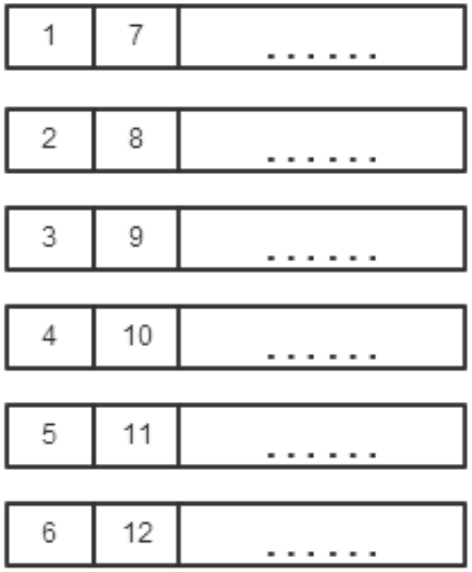
下标	1	3	4	2	0	5
对应关键字	a	a	a	b	d	d
分组号	0	0	0	1	2	2

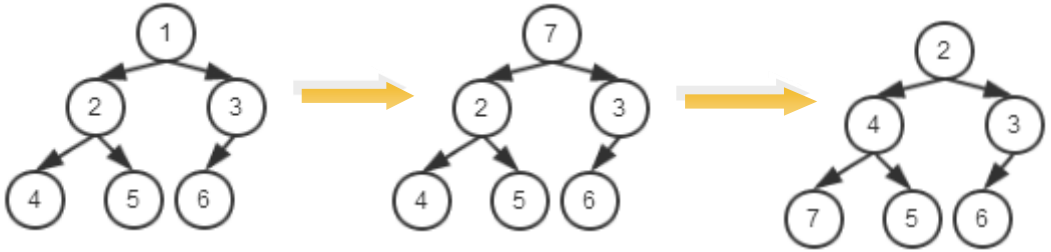
基于归并排序的分组

算法优化

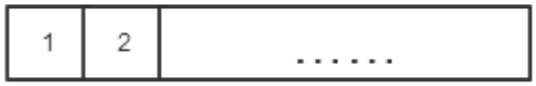
➢使用堆降低每趟归并的复杂度

- 将每个有序序列的首个元素作为节点建立小(大)顶堆,然后把堆顶写入结果集，并把相应序列的下一个元素替换堆顶元素，最后调整堆；
- 每一趟归并的复杂度由原来的O(b)降低为O(logb)，其中b是有序序列数目





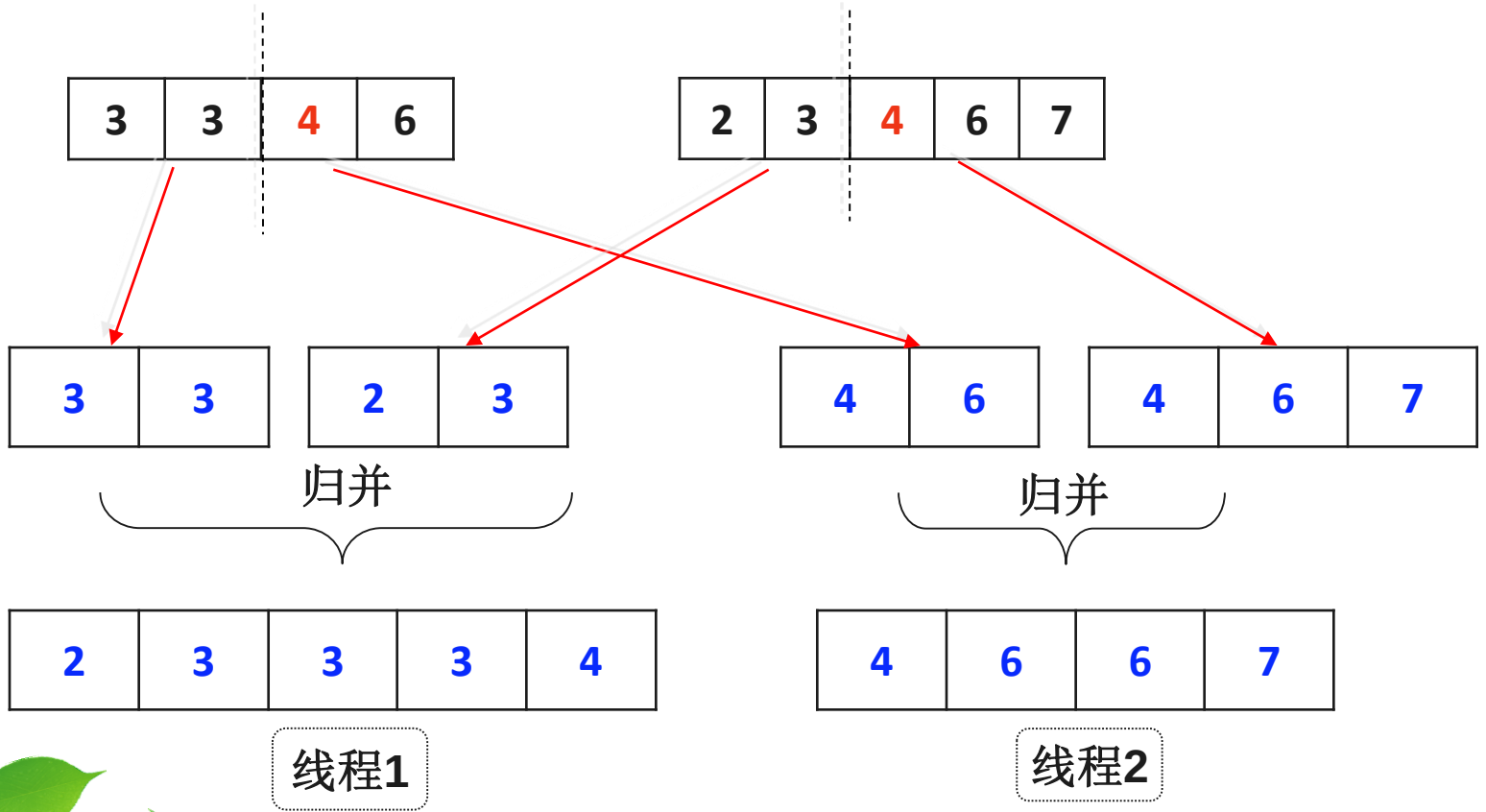
结果集：



基于归并排序的分组

归并过程并行化

选取若干个主元，用主元将每个有序序列分割成多个段；
第i个线程负责每个有序序列的第i段数据的归并。



分配分组号

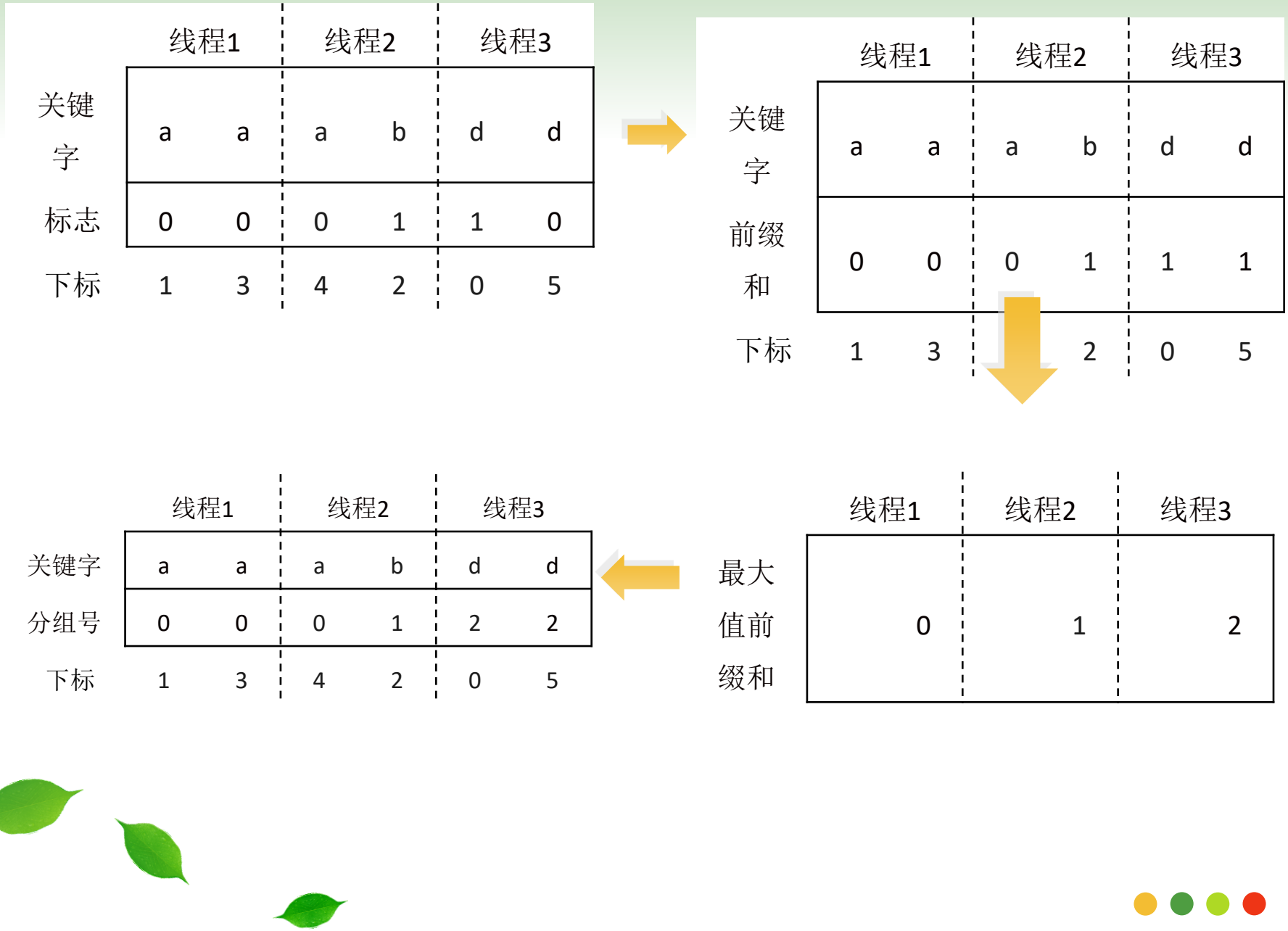
	线程1		线程2		线程3	
关键字	a	a	a	b	d	d
下标	1	3	4	2	0	5

✓线程i从第1个元素开始依次扫描子集合i，比较相邻两个元素的值，如果当前元素的值与前一个元素的值相等，则设置标志0，否则设置标志1

✓线程i对子集i求前缀和

✓主线程对各个子集最大值求前缀和

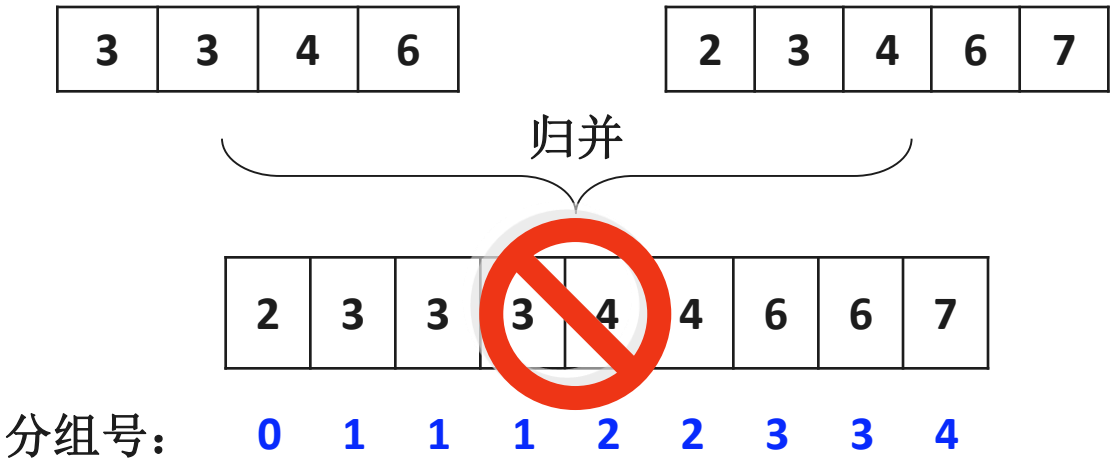
✓线程i将i-1的前缀和值加到分区中分组号


线程1
线程2
线程3
关键字
a a a b d d
标志
0 0 0 1 1 0
下标
1 3 4 2 0 5
线程1
线程2
线程3
关键字
a a a b d d
前缀和
0 0 0 1 1 1
下标
1 3 2 0 5
线程1
线程2
线程3
最大值前缀和
0 1 2
线程1
线程2
线程3
关键字
a a a b d d
分组号
0 0 0 1 2 2
下标
1 3 4 2 0 5

进一步优化

合并归并过程和分配分组号过程

把归并过程和分配分组号过程合并了，绕开了归并过程的写排序结果的环节。



算法改进前后的时间对比

实验测试

对4个分组数不同，元组数都是$1.073 * 10^9$的数据集的测试，使用12线程的归并排序比串行的qsort平均快了4倍；

基于归并排序的分组的性能是基于串行qsort的分组的性能的4倍，但平均耗费70s仍然比较长

数据集	分组数	qsort T1（s）	归并排序 T2（s）	T1/T2
DataSet1	16	194.65	57.16	3.41
DataSet2	32,768	277.96	66.81	4.16
DataSet3	1,048,576	313.55	72.72	4.31
DataSet4	33,554,432	348.42	81.64	4.27
平均值		283.65	69.58	4.04

主要内容

1. 基于归并排序的分组

- 改进归并过程，给出归并过程的并行化方案

2. 分段计数分组法

- 研究分段计数分组算法，提出分段计数分组法的并行化方案

3. 基于哈希表的分组

- 研究基于哈希表算法的分组，提出独立哈希表法的cache优化方案
- 在MIC平台上实现基于独立哈希表的分组，并进行向量化优化

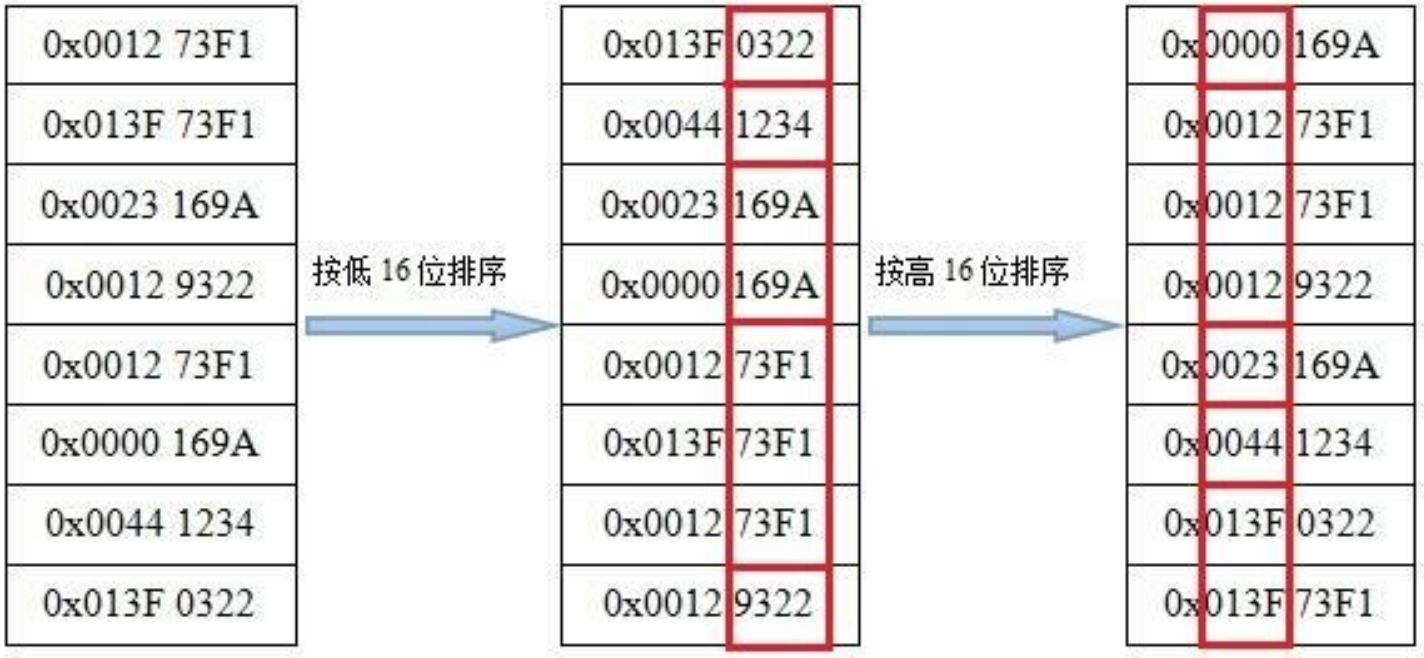
基于计数方式的分组

核心思想：对任意数据，其在内存中可以视为8、16、32位无符号整数的组合。如32位整数，其内存中的高16位和低16位均可以看作一个整数。对字符串，连续两个字符可以看作一个16位的整数。对浮点数，在将其转换为由数字组成的字符串后同样可以16位数的组合。由于计数排序在数字范围较小时具有非常高的性能(如16位整数)，我们根据这种特点提出分段计数分组算法。

分段计数分组算法

基本思想

从数据的低位到高位以**16bit**为单位对数据进行多轮计数排序。时间度为**O(n*d)**，其中**d**是计数排序趟次。



同时注意到一趟计数排序是一个分区过程

串行分段计数分组

(1)设辅助数组A保存待分组数据下标，设计数组cnt保存16位无符号整数落入0~2^{16}-1范围内数的个数；

(2)将待分组数据划分为16位无符号整数组合，分组过程中按高16位或低16位优先方式依次取数；

(3)将16位无符号整数在内存中完成计数，结果保存在cnt中，计算前缀和；

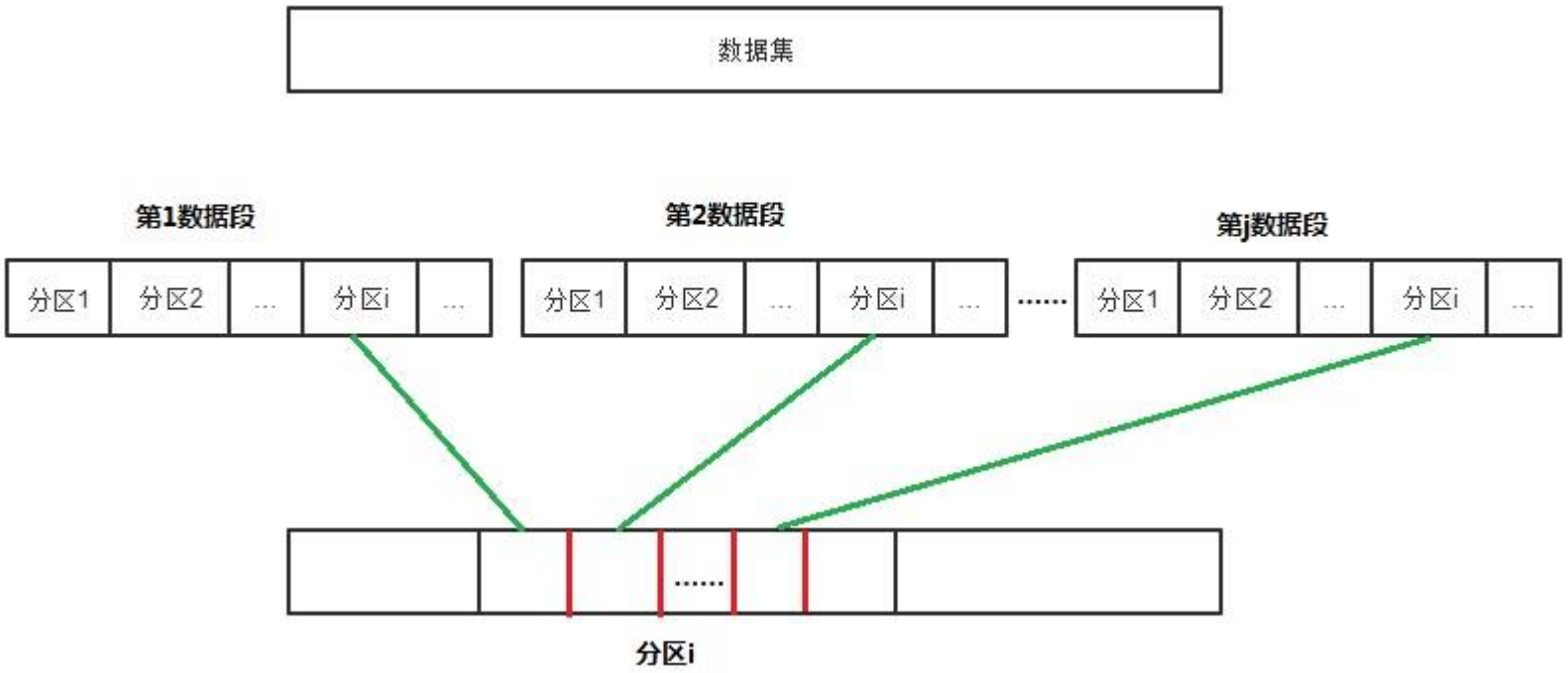
(4)根据cnt中的前缀和调整对应辅助数组A中下标的顺序，重复上述过程m趟后数组A中保存的下标即为分组后元素的下标；

(5)应用分配分组号算法完成分组

并行分段计数分组

计数排序并行化

将数据集平均划分成p个段，由p个线程分别进行计数排序，最后归并——对一趟计数排序，每个数据段最多产生65536个分区，用一个二维数组记录第j个数据段的第i个分区应该保存到结果集中的地址。



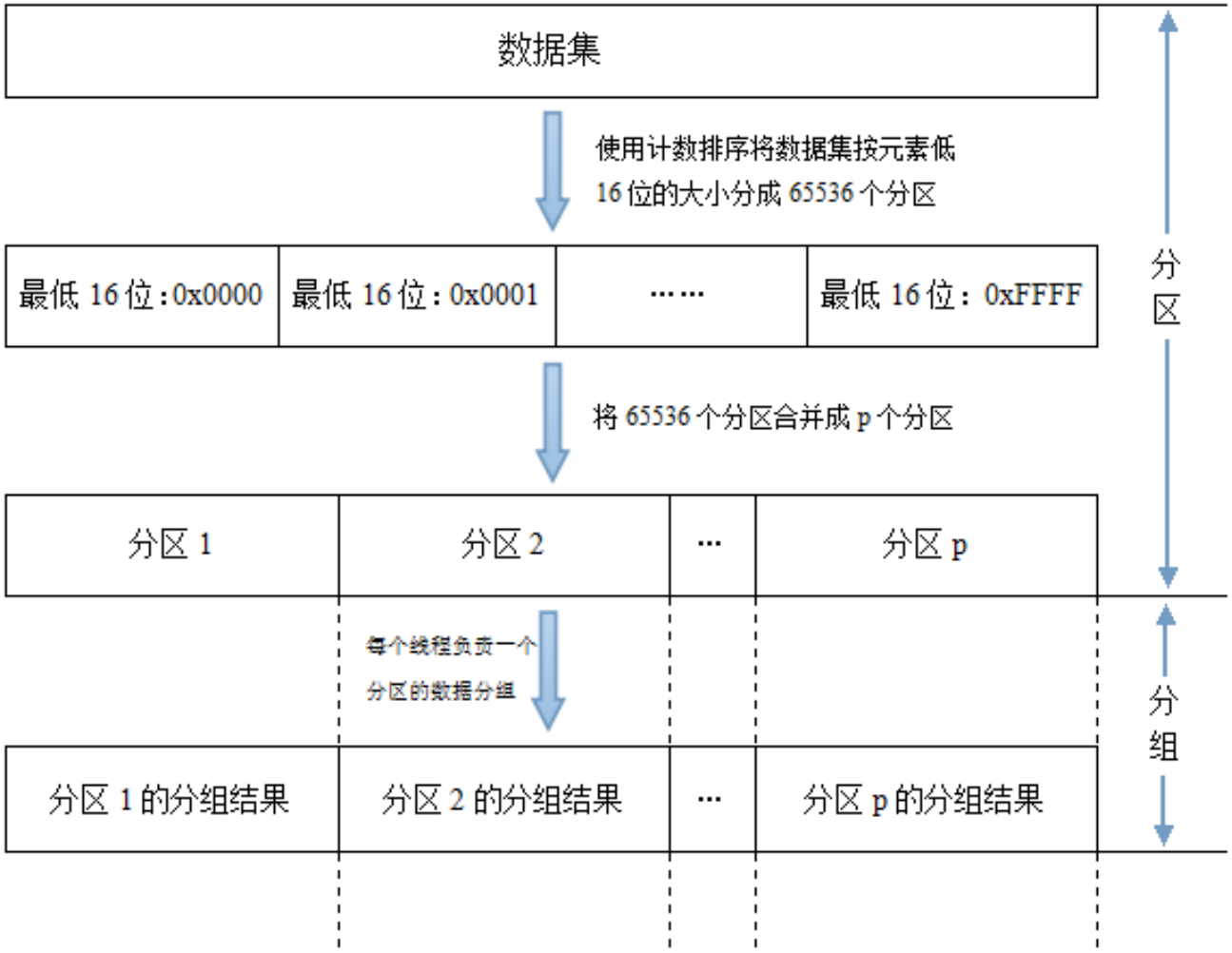
分段计数分组算法

总体过程

1、按照上述并行方式执行低16位的计数排序；

2、将第一轮排序产生的分区合并成p个；

3、一个分区内的数据由一个线程负责对其进行高位的计数分组。



第一趟并行计数分组

假设待分组数据集 S 的大小为 n(n>0)，辅助下标集合 I，可用线程数为 p (p>0)，第一趟排序的结果被保存到 TMP 空间中。第一趟排序的过程如下：

(1) 初始化一个二维数组 cnt[p+1][2^{16}]，把下标集 I 平均分为 p 份($I_0, I_1, ..., I_{p-1}$)，由线程号为 i(0≤i<p)的线程对子集 I_i 按元素低 16 位进行计数，并把计数结果保存在 cnt[i]中，如：

0	0	5	3	2	4	1	…
1	3	2	1	4	2	3	…
2	1	3	2	3	4	2	…
3	3	4	0	4	3	1	…
4	0	0	0	0	0	0	…

(2)分别对 cnt 数组的每个列的数值求和，并保存于 cnt[p]中。

(3)对数组 cnt[p]求前缀和.

0	0	5	3	2	4	1	…
1	3	2	1	4	2	3	…
2	1	3	2	3	4	2	…
3	3	4	0	4	3	1	…
4	7	14	6	13	13	7	…

0	0	5	3	2	4	1	…
1	3	2	1	4	2	3	…
2	1	3	2	3	4	2	…
3	3	4	0	4	3	1	…
4	7	21	27	40	53	60	…

(4) 从第 p-1 行开始，令 cnt[i-1] = cnt[i]-cnt[i-1]，直到第 0 行，cnt 的 1~p 行即为各线程对应低 16 位取值为 0~(2^{16}-1)的元素的存放位置。

0	0	7	21	27	40	53	…
1	0	12	24	29	44	54	…
2	3	14	25	33	46	57	…
3	4	17	27	36	50	59	…
4	7	21	27	40	53	60	…

(5) 线程并行扫描数据分区，按照步骤(4)计算得到的位置数组重排存放下标的辅助数组 I，结果存入在数组 TMP 中。

分段计数分组算法

实验测试

使用12个线程执行程序，测试算法对4个数据集分组的性能

数据集	分组数	串行时间	并行时间	加速比	T_{qsort}/ $T_{并行计数}$	$T_{并行归并}$/ $T_{并行计数}$
DataSet1	16	42.49	5.89	7.21	33.05	9.70
DataSet2	32,768	46.05	8.93	5.56	31.13	7.48
DataSet3	1,048,576	50.92	8.74	5.83	35.88	8.32
DataSet4	33,554,432	55.50	8.64	6.42	40.33	9.95
平均值		48.74	8.05	6.26	35.10	8.86

分段计数分组法的性能是串行的基于qsort的分组的35倍，是基于并行归并算法的9倍左右

主要内容

1. 基于归并排序的分组

- 改进归并过程，给出归并过程的并行化方案

2. 分段计数分组法

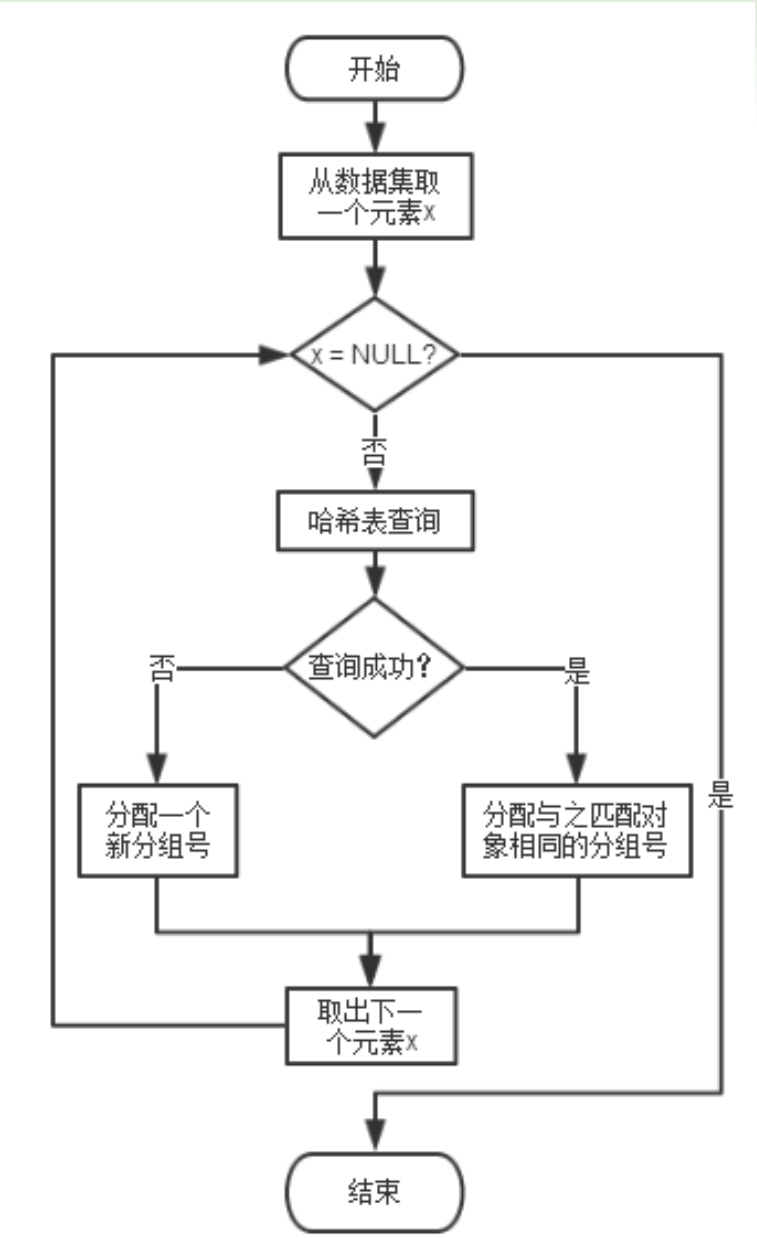
- 研究分段计数分组算法，提出分段计数分组法的并行化方案

3. 基于哈希表的分组

- 研究基于哈希表算法的分组，提出独立哈希表法的cache优化方案
- 在MIC平台上实现基于独立哈希表的分组，并进行向量化优化

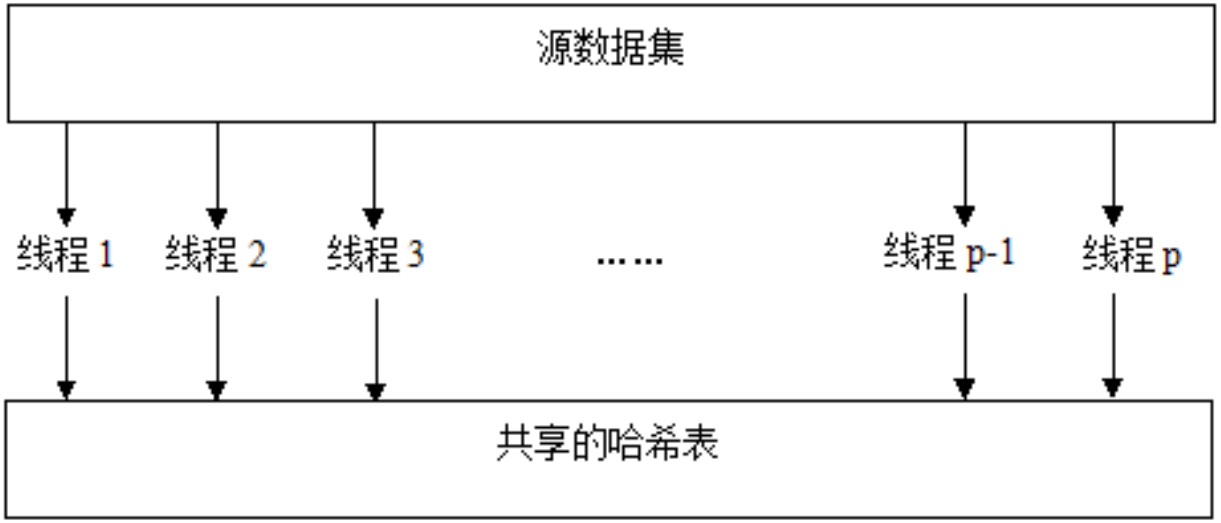
哈希分组

基本思想

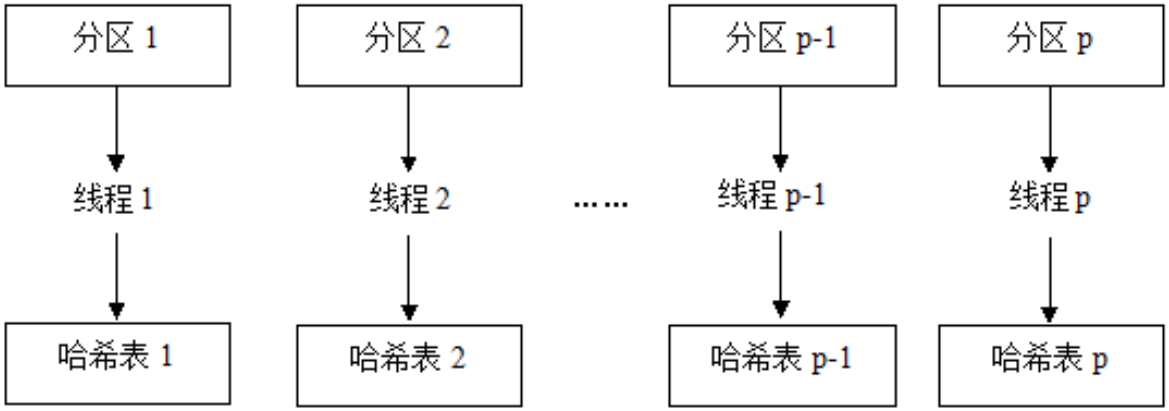

开始
从数据集取一个元素x
x = NULL?
否
哈希表查询
查询成功?
否
是
分配一个新分组号
分配与之匹配对象相同的分组号
是
取出下一个元素x
结束

哈希分组并行化

共享哈希表：所有的线程访问一个共享的哈希表。如果线程在哈希表查找成功，则直接为记录分配分组号，然后处理下一条记录；否则就“互斥”地将记录插入哈希表，并为记录分配一个新的分组号。



独立哈希表：将数据集 S 均匀分成 p 个分区，每个线程负责对一个分区进行数据分组，每个分区都对应一个独立的哈希表，各个线程执行的操作与用单个线程对数据分组的情况完全一样；最后，当所有线程完成对相应的分区的数据分组后，由主线程将各个分区的分组结果汇总。

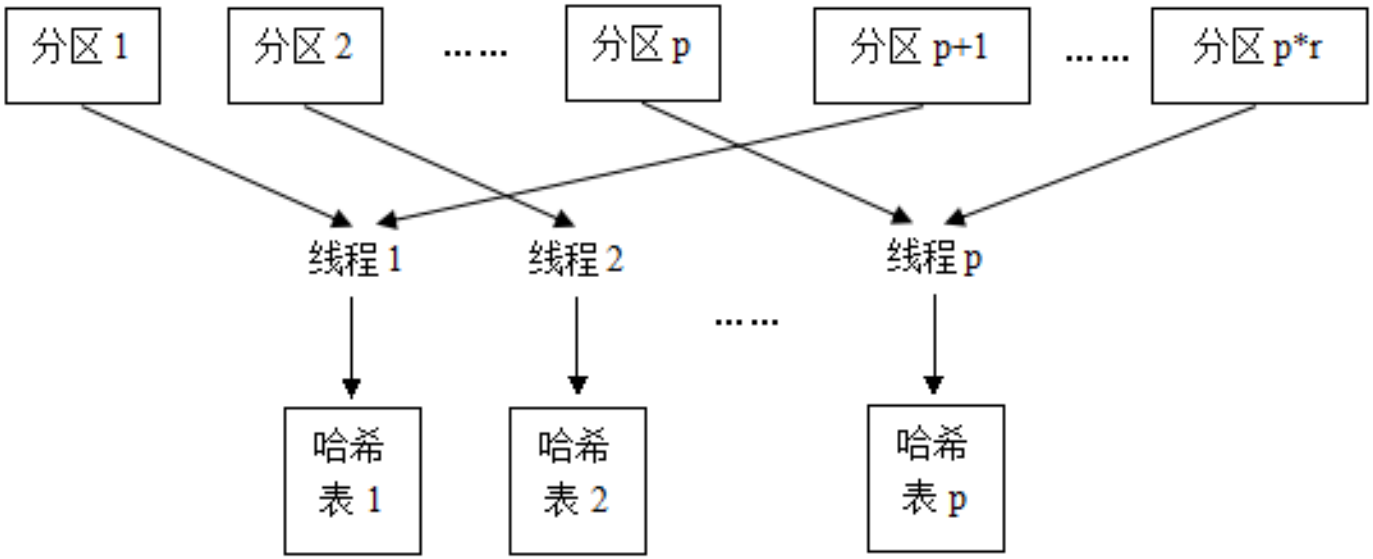


基于哈希表的分组

cache优化

存在问题： 哈希表被跳跃访问，访存行为不具备的时间局部性和空间局部性。

优化：根据cache大小降低分区粒度，使得在每个分区的分组过程中，被访问数据和哈希表基本常驻在cache



假设需要分组的数据集 S，由 p 个线程处理数据分组，cache 大小为 M。如果每个线程在一个时间段内只访问少数几个连续的内存空间且总大小不超过$\lfloor M/p \rfloor$，那么程序会有很可观的访存局部性。在数据分组过程中，每个线程访问的数据包括源数据、分区结果、哈希表和分组结果等，经过估算，如果将源数据划分成大小为 $z=\frac{1}{5}\lfloor M/p \rfloor$的分区，就很有可能大大改善程序访存的局部性。数据分组过程如下：

(1)将源数据分成大小为 $z=\frac{1}{5}\lfloor M/p \rfloor$的分区；

(2)分配 p 个大小为 z 的哈希表。各个线程迭代地将还未处理的分区进行数据分组，每个线程只访问对应的哈希表；

(3)将各个分区的分组结果汇总，得到最终的分组结果。

基于哈希表的分组

MIC实现和向量化优化

1、在MIC平台上实现

2、向量化：MIC处理器支持512位宽的Knights Corner指令，通过使用knight corner指令使程序同时对16个记录进行哈希表查询

x16	…	…	…	…	…	…	…	…	…	…	x5	x4	x3	x2	x1

y16	…	…	…	…	…	…	…	…	…	…	y5	y4	y3	y2	y1

x16 op y16	…	…	…	…	…	…	…	…	…	…	x5 op y5	x4 op y4	x3 op y3	x2 op y2	x1 op y1

基于哈希表的分组

实验测试

对4个数据集测试算法的性能，在CPU平台使用12条线程测试，在MIC平台使用200条线程测试

数据集	分组数	串行	并行	cache优化	mic实现	向量化
DataSet1	**16**	17.54	12.00	13.14	10.62	7.64
DataSet2	**32,768**	18.61	13.63	12.24	7.20	7.02
DataSet3	**1,048,576**	21.74	17.84	14.26	7.67	7.04
DataSet4	**33,554,432**	31.35	22.78	17.82	8.78	7.72

综合对比

综合对比

对4个数据集测试算法的性能，在CPU平台使用12条线程测试，在MIC平台使用200条线程测试

数据集	基于串行快速排序的分组算法	基于归并排序的并行分组算法		并行分段计数分组算法		多核CPU上基于独立哈希表并行分组算法		MIC平台上基于独立哈希表并行分组算法	
		耗时	提高倍数	耗时	提高倍数	耗时	提高倍数	耗时	提高倍数
DataSet1	194.65	57.16	3.41	5.89	33.05	13.14	14.81	7.64	25.48
DataSet2	277.96	66.81	4.16	8.93	31.13	12.24	22.71	7.02	39.60
DataSet3	313.55	72.72	4.31	8.74	35.88	14.26	21.99	7.04	44.54
DataSet4	348.42	81.64	4.27	8.64	40.33	17.82	19.55	7.72	45.13