分布式统计计算

7. 常见并行算法的实现

李舰

华东师范大学

2018-01-07

目录

1. 排序和归并
   - 冒泡排序
   - 快速排序
   - 并行的桶排序
2. 大规模矩阵运算

目 录

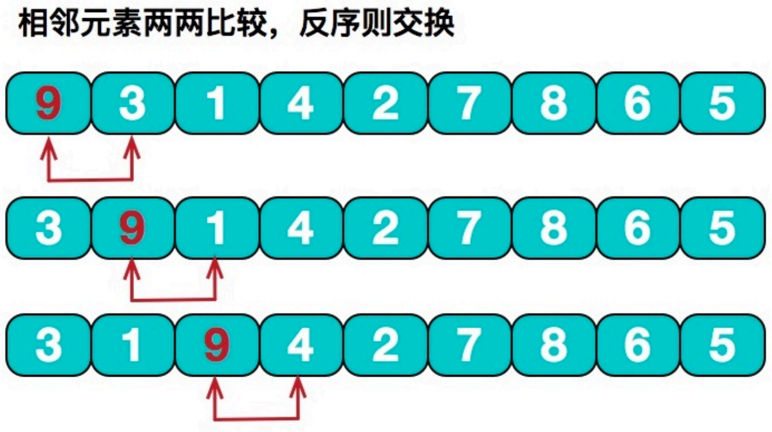
1 排序和归并
- 冒泡排序
- 快速排序
- 并行的桶排序

2 大规模矩阵运算

冒泡排序算法

- 算法思路
  - 对相邻的元素进行两两比较，顺序相反则进行交换。
  - 每一次迭代会将最大的元素浮到顶端，最终达到完全有序。
  - 每一次迭代后，需要排序的元素会减少一个，直到最终结束。
- 时间复杂度
  - 如果完全遍历的话，需要两两比较的次数为 $(n-1)+(n-2)+\cdots+1$，化简后得 $(n-1)n/2$，也就是 $O(n^2)$。
  - 如果设置停止条件，确认有序后不再迭代，这样最理想的情况下仅需 $n-1$ 次就能完成，但平均来看，不影响该算法是 $O(n^2)$ 复杂度。
  - 冒泡排序是一种很简单的排序算法，经常拿来当反面教材，因为时间复杂度比较高。

算法介绍（I）



算法介绍（II）


第一轮完毕，将最大元素9浮到数组顶端
3 1 4 2 7 8 6 5 9
同理,第二轮将第二大元素8浮到数组顶端
1 3 2 4 7 6 5 8 9
排序完成
1 2 3 4 5 6 7 8 9

冒泡排序函数

- 代码说明
  - 定义函数 sort1，输入待排序的向量 x，输出一个排好序的向量。与 R 内置的 sort 函数用法类似。
  - 每一步迭代和交换位置通过 R 语言的循环与向量来实现，性能不是很好。
- 代码示例

```
sort1 <- function(x) {
  n <- length(x)
  if (n <= 1) return(x)
  for (i in 1:(n-1)) {
    for (j in 1:(n-i)) {
      if (x[j] > x[j+1]) {
        x[j:(j+1)] <- x[(j+1):j]
        if (n == 2) return(x)
      }
    }
  }
  return(x)
}
```

目 录

1. 排序和归并
   - 冒泡排序
   - 快速排序
   - 并行的桶排序
2. 大规模矩阵运算

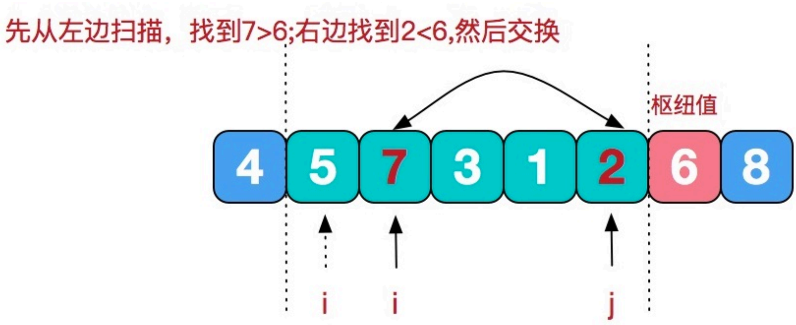
快速排序算法

- 算法思路
  - 首先确定一个主元（Pivot），通常取第一个数。也可以同时取三个，首尾各一，加上中间一个，把中间的当作主元。
  - 目标是让主元左边的数都比它小，右边的数都比它大，因此可以从左遍历找小的，从右遍历找大的，然后交换，这是算法比较快的关键。
  - 对于左段和右段，通过递归来继续计算。
- 时间复杂度
  - 该算法中数据会分成 2 份、4 份、8 份，以此类推，最后所有的组大小都为 1，这个过程大约需要 $log(n)$ 步。
  - 每一步中，将所有元素和主元比较，大约需要 n 个操作。
  - 可得时间复杂度为 $O(nlog(n))$。

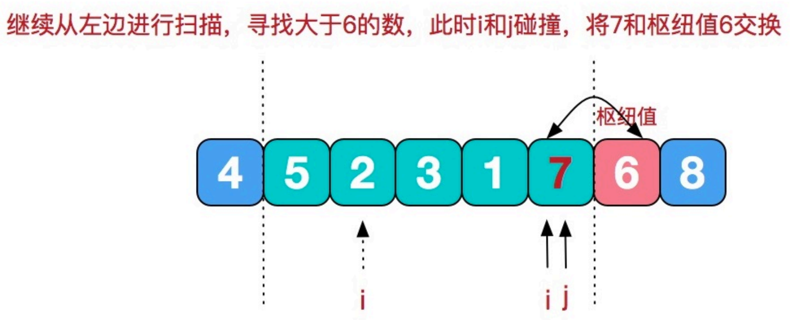
算法介绍（I）


0 1 2 3 4 5 6 7
原始数组
4 5 7 8 1 2 3 6
0 1 2 3 4 5 6 7
对这三个值进行排序
4 5 7 6 1 2 3 8
0 1 2 3 4 5 6 7
将枢纽值6放在数组末尾
4 5 7 3 1 2 6 8

算法介绍（II）



算法介绍（III）



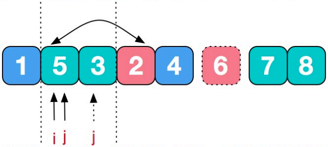
算法介绍（IV）


此时第一轮分割完成，我们可以看到，左边均小于6，右边均大于6
枢纽值
4 5 2 3 1 6 7 8
递归对子序列进行这种处理（先三位取中，再以中值分割）
4 5 2 3 1
7 8

算法介绍（V）

对左序列三数取中，并将中值放置数组末尾，然后扫描分割，右序列同理

依然从左边开始扫描 找到5>2,然后从右边扫描，没找到小于2的数，但此时i和j碰撞，此轮结束，交换5和2



此时，枢纽值2将左子序列分成两部分，左边{1}均小于2，右边{3,5,4}均大于2。右子序列同样处理，此处不表。

然后继续递归处理，对每个子序列先进行三数取中，在以中值进行分割，最终使得整个数组有序。

快速排序函数

- 代码说明
  - 定义函数 sort2，输入待排序的向量 x，输出一个排好序的向量。与 R 内置的 sort 函数用法类似。
  - 主要演示递归的实现方式。省略了具体每一步迭代换位的过程，改用 R 的向量操作来实现。
- 代码示例

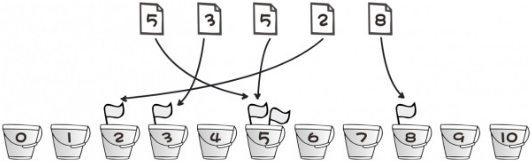
```
sort2 <- function(x) {
  if(length(x) <= 1) return(x)
  x2 <- x[1]
  x1 <- x[x < x2]
  x3 <- x[x > x2]

  y1 <- sort2(x1)
  y2 <- x2
  y3 <- sort2(x3)
  y <- c(y1, y2, y3)
  return(y)
}
```

目 录

1. 排序和归并
   - 冒泡排序
   - 快速排序
   - 并行的桶排序
2. 大规模矩阵运算

桶排序算法

- 算法思路
  - 将数据分成不同的“桶”，每个桶包含部分数据。
  - 对每个桶的数据进行排序，得到多个有序向量。
  - 将多个有序向量合并成一个更大的有序向量，称为“归并”。
- 并行的桶排序
  - 为了归并简单，可以将数据根据分位数来划分，这样确保每个“桶”排序后的结果可以直接拼接到一起。
  - 如果每个“桶”的数据随机划分，需要使用算法进行归并，比如树方法。
- 注意
  - “桶排序”也指另外一种将数据填入编好号的桶中的算法。



并行桶排序函数

- 并行代码

```
mcbsort <- function(x, ncores = 2, nsamp = 1000) {
  require(parallel)
  samp <- sort(x[sample(1:length(x), nsamp,
    replace = TRUE)])
  dowork <- function(me) {
    k <- floor(nsamp / ncores)
    if (me > 1) mylo <- samp[(me -1) * k + 1]
    if (me < ncores) myhi <- samp[me * k]
    if (me == 1) {
      myx <- x[x <= myhi]
    } else {
      myx <- x[x > mylo & x <= myhi]
    }
    sort(myx)
  }
 res <- mclapply(1:ncores, dowork, mc.cores = ncores)
  c(unlist(res))
}
```

并行桶排序函数

- 测试代码

```
test <- function(n, ncores) {
  x <- runif(n)
  mcbsort(x, ncores = ncores, nsamp = 1000)
}
test(100, 2)
```

- 代码说明
  - 定义函数 mcbsort，使用 parallel 包里的 multicore 来实现并行。参数 x 表示待排序的向量，参数 ncores 表示核心数，参数 nsamp 表示抽取子样本的数目。
  - 函数 test 用来产生随机数并调用 mcbsort 进行排序，返回 mcbsort 的结果。
  - 由于该示例使用了 multicore 的功能来并行，所以只能运行在类 Unix 的操作系统中。
  - 该函数主要演示并行，对于每个节点的排序算法直接调用 sort 实现。

目 录

1 排序和归并

2 大规模矩阵运算
- 示例：图的连通性
- 矩阵乘法的并行
- 其他并行矩阵运算

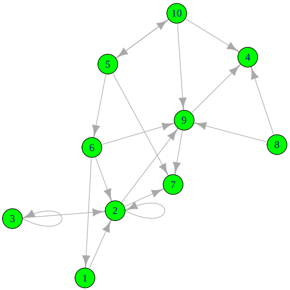
目　录

1 排序和归并

2 大规模矩阵运算
- 示例：图的连通性
- 矩阵乘法的并行
- 其他并行矩阵运算

连通图

- 连通图的概念
  - 假设图中共有 n 个顶点，如果顶点 i 到 j 之间存在一条边，则称这两个顶点是邻接的。
  - 如果顶点 i 可以在 k 步之内到达顶点 j，则称这两个点是连通的，如果图中任意两个顶点之间是连通的，则称为连通图（Connected Graph）。
- 连通图示例
  - 假设有向图中共有 10 个点，其关系如图所示。



邻接矩阵

- 连通图和邻接矩阵
  - 定义邻接矩阵 A，如果顶点 i 到 j 之间存在一条边，则元素 $a_{ij}=1$。$R^{(k)}$ 表示一个 0-1 矩阵，其中元素 $r_{ij}^{(k)}$ 表示可以在 k 步之内从 i 到 j，该矩阵称为可达矩阵。
- 邻接矩阵示例

$$A=\begin{bmatrix}0&1&0&0&0&0&0&0&0&0\\0&1&0&0&0&0&1&0&1&0\\0&1&1&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&1&1&0&0&1\\1&1&0&0&0&0&0&0&1&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&1&0&0&0&0&1&0\\0&0&0&1&0&0&1&0&0&0\\0&0&0&1&1&0&0&0&1&0\end{bmatrix}$$

连通性的计算

- 总体连通性
  - 用 R 表示图中的连通性，有：

$$R = b(\sum_{k=1}^{n-1} R^{(k)})$$

  - 其中函数 $b()$ 表示一个布尔操作，将非零的数字变成 1，0 仍然是 0。
- 计算可达矩阵
  - 假如我们要计算 $R_{2,4}^{(2)}$，即考虑经过 2 步从顶点 2 到顶点 4 的情况，可能包括顶点 2 到 1 再到 4、顶点 2 到 2 再到 4、顶点 2 到 3 再到 4 等情况。
  - 用公式描述可得：
    $R_{2,4}^{(2)} = a_{2,1} * a_{1,4} + a_{2,2} * a_{2,4} + a_{2,3} * a_{3,4} + \cdots + a_{2,10} * a_{10,4}$，
    可以发现对应着矩阵乘法运算。由此可得：

$$R^{(k)} = b(A^k)$$

  - 因此图形连通性的问题可以转化成矩阵求幂的问题。

矩阵求幂问题的简化

- Log 技巧
  - 对于 A^k 运算，我们需要计算 $k-1$ 次矩阵乘法。
  - 我们可以使用 Log 技巧，比如计算 A^{16}，可以将其平方得到 A^2，再平方得到 A^4，再做两次平方分别得到 A^8 和 A^{16}，这样只需要 4 次矩阵相乘即可。
  - 在图的连通性问题中，我们需要求 $n-1$ 次幂，我们可以使用这种方式将计算次数减小到 $2^{\lceil \log_2 n-1 \rceil}$。
- 矩阵分解
  - 矩阵 A 可能可以对角化，也就是说可以找到矩阵 C，使得 $A = C^{-1}DC$，其中 D 是一个对角矩阵，其元素是 A 的特征值。可以使用计算特征值的算法进行分解。
  - 那么有 $A^k = C^{-1}D^kC$，其中 D^k 非常容易计算。
- 并行计算
  - 矩阵乘法是最常用的矩阵运算方式，如果能通过并行计算实现，可以大大地提升运算效率。
  - 很多矩阵运算技巧都用到了矩阵乘法的并行算法。

目 录

1 排序和归并

2 大规模矩阵运算
- 示例：图的连通性
- 矩阵乘法的并行
- 其他并行矩阵运算

矩阵的分块乘法

- 分块乘法公式
  - 假设矩阵 A 和 B 是两个可以相乘的矩阵，我们将他们分别分解成 $s\times t$ 和 $t\times r$ 个块：

$$A=\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1t} \\ A_{21} & A_{22} & \dots & A_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ A_{s1} & A_{s2} & \dots & A_{st} \end{bmatrix}, B=\begin{bmatrix} B_{11} & B_{12} & \dots & B_{1r} \\ B_{21} & B_{22} & \dots & B_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ B_{t1} & B_{t2} & \dots & B_{tr} \end{bmatrix}$$

  - 其中 $A_{ik}\quad (k=1,2,\dots,t)$ 的列数分别等于 $B_{kj}\quad (k=1,2,\dots,t)$ 的行数。则有：

$$AB=\begin{bmatrix} C_{11} & C_{12} & \dots & C_{1r} \\ C_{21} & C_{22} & \dots & C_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ C_{s1} & C_{s2} & \dots & C_{sr} \end{bmatrix}$$

  - 其中 $C_{ij}=\sum_{k=1}^{t} A_{ik}B_{kj}\quad (i=1,\dots,s;j=1,\dots,r)$。

矩阵乘法并行：Snowdoop 方法

- 代码简介
  - 假如内存不够（比如使用 GPU），可以分块后相乘。
  - 将矩阵 a 分成 ntiles 块，分别和矩阵 x 相乘。
- 代码示例

```
biggpuax <- function(a, x, ntiles) {
  require(parallel)
  require(gputools)
  nrx <- nrow(a)
  y <- vector(length = nrx)
  titlesize <- floor(nrx / ntiles)
  for (i in 1:ntiles) {
    tilebegin <- (i-1) * tilesize + 1
    tileend <- i * tilesize
    if (i == ntiles) tileend <- nrx
    tile <- tilebegin:tileend
    y[tile] <- gpuMatMult(a[tile,,drop=FALSE],x)
  }
  return(y)
}
```

矩阵乘法并行：消息传递机制

- **思路简介**
  - 如果矩阵足够大，需要进行分布式存储（比如 Hadoop 文件系统），因此计算时需要分块，最后的乘积也会分布式存储。
  - 常用 Fox 算法，其实就是矩阵分块乘法的实现。
- **伪码简介**
  - 假设 $\sqrt{p}$ 能被 n 整除，从而将每个矩阵分成 $\sqrt{p} \times \sqrt{p}$ 规模的分块，每个矩阵被分成 m 行和 m 列的块，其中 $m = n/\sqrt{p}$。
  - 负责计算 C_{ij} 的节点的操作如下所示（同时其他节点操作其自己的 i 和 j）：

```
iup = i+1 mod m;
idown = i-1 mod m;
for (k = 0; k < m; k++) {
  km = (i+k) mod m;
  把 A[i, km] 广播到处理 C 的第 i 行的所有节点
  C[i, j] = C[i, j] + A[i, km] * B[km, j]
  把 B[km, j] 发送给处理 C[idown, j] 的节点
  从处理 C[iup, j] 的节点接收 B[km+1 mod m, j]
}
```

矩阵乘法并行：多核机器

- 思路简介
  - 首先将矩阵分块乘法写成循环的形式，在执行最内层循环（k）的时候，遍历 A 的一行和 B 的一列。
  - 可以利用 OpenMP 的机制来把 i 层循环和 j 层循环并行化。
- 伪码简介
  - 使用 OpenMP 中的 `for pargma` 语句来实现：

```
for (i = 0; j < nrowsa; i++) {
  for (j = 0; j < ncolsb; j++) {
    sum = 0;
    for (k = 0; k < ncolsa; k++) {
      sum += a[i][k] * b[k][j];
    }
    c[i][j] = sum;
  }
}
```

  - 这并行化了外层循环，我们用 p 代表线程数，由于循环的每个 i 都处理了 A 中的一行，我们这里将 A 按行分成了 p 份，一个线程会计算 A 中的一行和整个 B 的乘积，得到了 C 中对应的一行。

矩阵乘法并行：GPU 计算

- 思路简介
  - GPU 计算中内存是主要问题，如果需要进行相乘的矩阵还没有在设备（显存）中，需要被复制过去，通过分块矩阵计算。
- 伪码简介
  - CUDA 在使用大量线程的时候工作得更好，可以让每个线程计算乘积 C 的一个元素：

```
__global__ void matmul(float *a, float *b, float *c,
  int nrowsa, int ncolsa, int ncolsb)
{
  int k, i, j; float sum;
  // 这个线程会计算 c[i][j];
  // i 和 j 的值由线程和 block ID 决定
  sum = 0;
  for (k = 0; k < ncolsa; k++) {
    //将 a[i,k]*b[l,j] 加到 sum 上
    sum += a[i*ncolsa + k] * b[k*ncolsb + j];
  }
  //对 c[i,j] 进行赋值
  c[i*ncolsb + j] = sum;
}
```

目 录

1 排序和归并

2 大规模矩阵运算
- 示例：图的连通性
- 矩阵乘法的并行
- 其他并行矩阵运算

矩阵运算并行工具

- BLAS 函数库
  - OpenBLAS 是已经停止的 GotoBLAS 项目的继续，其主页为 `http://www.openblas.net/`。
  - 在其他可以并行的框架中也存在可以直接使用的 BLAS，比如 Spark 平台、GPU 平台、其他商业平台等。
- 其他矩阵运算并行算法
  - 线性系统求解，可以使用高斯消去法和 LU 分解，还可以使用经典的 Jacobi 算法，可以用来求解线性方程组。
  - QR 分解将一个矩阵分解成一个标准化的正交矩阵 Q 和一个上三角矩阵 R 的乘积，也是常用的矩阵运算方法，很难并行化，但是容易求得近似解。
  - 特征分解又称为谱分解，也是常用的矩阵运算方法。设 A 是 $k \times k$ 对称矩阵，其第 i 个特征值记为 λ_i，第 i 个特征向量记为 e_i，则有 $A = \sum_{i=1}^{k} \lambda_i e_i e_i'$。
  - 设 A 是 $m \times k$ 的矩阵，则存在一个 $m \times k$ 的矩阵 U，一个对角线为向量 d 的对角矩阵 D，一个方阵 V，使得 $A = UDV'$，称为奇异值分解。

Thank you!

李舰 Email: jian.li@188.com