南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

并行程序设计

KMeans 算法 SIMD 并行实验

田翔宇 2011748

年级：2020 级

专业：计算机科学与技术

摘要

在 ARM 与 x86 平台开展 Kmeans 的 SIMD 并行化实验。

实验代码已上传 GitHub。

关键字：KMeans SIMD 并行程序设计

目录

一、 问题重述

- K-Means 是发现给定数据集的 K 个簇的聚类算法, 之所以称之为 K-均值, 是因为它可以发现 K 个不同的簇, 且每个簇的中心采用簇中所含值的均值计算而成。
- 划分 K 个集合的过程中，设计距离的计算大量且重复，数据之间没有依赖关系。
- 计算过程主要包括：
  1. 计算各个点到各个质心之间的距离
  2. 选择距离最近的质心，将各个点划分到其所属的集合中
- 数学公式：
  - 对于给定的 n 组数据 $\{x_1, x_2, \cdots, x_n\}$,每个数据属性的维度均为 d 维 $x(a_1, a_2, \cdots, a_d)$，通过计算数据之间的相似度，将这 n 个数据划分成为 K 组。假设这 K 组分类记为 $S = \{s_1, s_2, \cdots, s_k\}$，$K$ 组分类中心的集合记为 $C = \{c_1, c_2, \cdots, c_k\}$。
  - 对于第 i 个节点，计算这个节点到第 k 个质心的距离

$$L_{ik} = \sqrt{\sum_{j=1}^{d}(x_i(a_j) - c_k(a_j))^2} \quad , 1 \leq k \leq K$$

  - 对于第 i 个节点，选择距离他最近的质心，并将其划分如该质心所属集合中

$$\min\{L_{ik}\}, 1 \leq k \leq K$$

二、 实验环境

环境如下：

- 平台：x86
- CPU 型号：Intel(R) Core(TM) i7-10710U
- CPU 频率：1.10GHz
- L1 cache：384KB
- L2 cache：1.5MB
- L3 cache：12.0MB

三、 实验设计

设计思路：

- KMeans 算法每次迭代中, 需要重复计算各个数据点到所有质心的距离, 而这个过程大量且重复, 且在不同数据点之间是数据没有依赖性, 十分适合进行向量化的处理。因此对 KMeans 的并行优化应从对各个数据点计算其到所有质心的距离这一过程开始。
- 并行计算：

  - 对于计算距离，采用向量化的手段处理：
    * 每次取出多个数据点同一维度的数据存储到向量寄存器当中
    * 造某一个质心同一维度的一个向量寄存器
    * 计算出这多个数据点和该质心在这一维度上的距离差，然后将这个距离差利用向量对位乘法相乘，得到平方距离
    * 循环重复上述操作，并累加这个距离，直到将多个数据点各个维度的距离全部计算累加完成
  - 为了能够让向量寄存器可连续取多个数据，我们考虑将存储数据点和各个维度数值的矩阵进行转置。在取出多个数据点同一维度的数据时，就不需要进行数据拼接，而是直接传入起始元素的地址即可

- 缓存优化：
  - 访问时是按照列去访问的，即当前维度的数据计算完之后，将会到下一行去取下一个维度的数据重复计算操作。当数据规模较大时，会导致大量 cache miss，因此会产生很多访存操作，极大影响并行算法的优化效果。
  - 为了提高命中，减少访存导致的额外开销，改变循环的顺序，最外层循环是对维度的迭代，内层循环是对数据的迭代。这样可以使得内层循环在每一步中都是相邻的移动，不会出现跨行的数据读取，这样就会提高数据的 cache 命中率。

- 内存对齐：
  - 向量存取时如果内存不对齐，原始内存不是按照存取规模的大小内存对齐的，则需要访存多次，再进行数据的拼接，极大影响效率。
  - 对并行算法进行内存对齐，将数据规模全部设置为 4 的次幂。
    * 判断存放数据数组的每一行的首地址是否是对齐的
    * 若对齐，则不处理；不对齐，则对未对齐元素进行串行化处理，同属对后续起始元素进行串行化处理
    * 起始串行处理之后，还可能会导致每一行剩余的元素不足，要对每一行剩余元素做串行处理

四、 实验结果分析

(一) ARM

实验要考虑的影响因素：

- 数据点的数量 N
- 数据的维度 D

故考虑使用控制变量进行结果分析。为了方便实验进行以及数据的采集，数据规模使用 4 的 n 次幂，数据维度使用 4 的 d 次幂表示。

1. 数据规模

选定数据维度为 16 时，各种优化算法表现效果随数据规模 n 的变化趋势如表1所示。

表 1: 相同数据维度不同数据规模

n	d	serial	parallel	parallel_cache	parallel_aligned
6	2	2.129	1.960	2.409	1.831
7	2	8.635	7.401	8.310	7.323
8	2	90.747	41.021	37.076	38.134
9	2	356.102	165.087	150.671	161.321
10	2	1348.531	579.992	468.571	560.141

根据数据计出加速比，如图1所示。

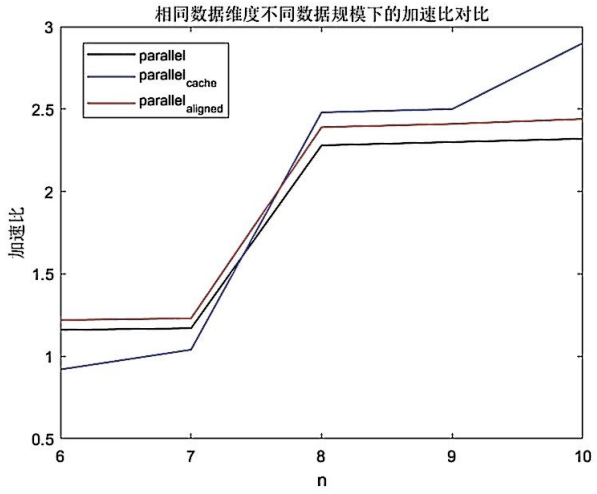

图 1: 相同数据维度不同数据规模

分析:

- 在较小数据规模下，串行算法的优化效果并不明显，原因可能是为算法虽然能够同时并行多条指令，但是每一条指令所消耗的时间加长
- 数据规模达到 4 的 8 次幂以上时，三种并行优化算法都能够取得明显的优化效果
- 未进行 cache 优化的算法，一次可以取出多条数据进行计算，所需要的周期数自然要少于串行算法，其中内存对齐的算法由于不涉及到数据的拼接，所需要的指令数会略低于内存不对齐的算法。
- 利用了 cache 优化的算法在数据规模增大时，优化效果最好，可见 cache 优化的重要性

2. 数据维度

选定数据规模为 4 的 8 次幂时，各种优化算法表现效果随数据规模 d 的变化趋势如表2所示。

表 2: 相同数据规模不同数据维度

n	d	serial	parallel	parallel_cache	parallel_aligned
8	2	89.98	38.43	36.98	35.32
8	3	421.43	175.23	145.12	167.67
8	4	3621.31	1030.43	572.37	955.83
8	5	18579.78	5189.64	2521.29	4800.92
8	6	70901.83	17923.54	9530.12	15420.02

根据数据计出加速比，如图2所示。

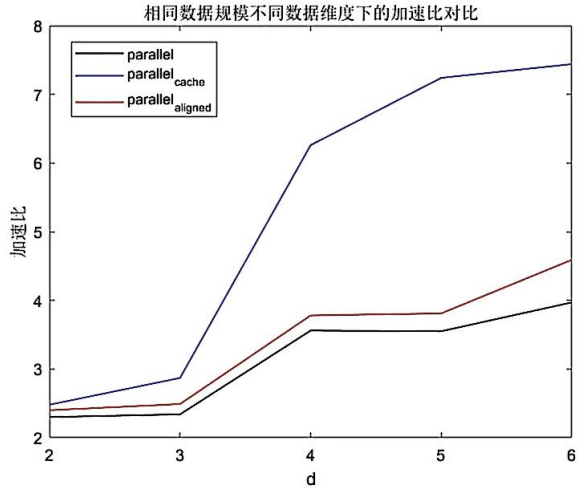

图 2: 相同数据规模不同数据维度

分析：

- 选定的数据规模较大，在较小的数据维度下，三种并行优化算法都能够取得不错的优化算法
- 随着数据维度的增加，未采用 cache 优化的算法增势与增幅大致相似数据维度较大时加速比接近 4，因为是四路展开
- 采用 cache 优化的算法的加速比远超 4，cache 优化能够极大的提高数据在缓存中的命中率。
- 实验中行存储 N 个数据同一维度的数据，列存储同一个数据点不同维度的值。当数据规模与数据维度都较大的时候，每一次访问下一行时，都可能导致 cache miss，而 cache 优化按照行进行计算的，能够有效降低 miss

(二) x86

在 x86 平台上，使用 SSE、AVX 和 AVX512 这三种并行指令集架构，编写了不同的串行算法和三种并行算法，对比了这三种架构下内存对齐对于并行算法的优化效果的影响。

实验设计思路：

- 与 ARM 平台一致，即采用向量化的存取手段：

- 每次取出多个数据点同一维度的数据存储到向量寄存器当中
- 造某一个质心同一维度的一个向量寄存器
- 计算出这多个数据点和该质心在这一维度上的距离差，然后将这个距离差利用向量对位乘法相乘，得到平方距离
- 循环重复上述操作，并累加这个距离，直到将多个数据点各个维度的距离全部计算累加完成

为了便于对比，选择数据规模为 4 的 6 次幂，数据维度为 4 的 8 次幂，各种指令集架构下的并行算法优化效果如下表3所示。

表 3: 不同指令集并行算法性能对比

serial		SSE_aligned	AVX_aligned	AVX512_aligned
15780.6	time	4680.2	2591.6	1526.1
	s/p	3.37	6.09	10.34
		SSE_unaligned	AVX_unaligned	AVX512_unaligned
	time	4832.9	2806.4	1729.3
	s/p	3.27	5.62	9.13
	unalign/align	1.03	1.08	1.13

分析：

- 选定的数据规模和数据维度较大，并行优化算法都能够取得不错的优化算法
- 采用向量化的优化，能够一次性处理多条数据，当数据规模和数据维度足够大的时候理论加速比应该逼近其向量寄存器一次性能够处理的数量
- 内存对齐的算法，优化效果并不明显，但也能看出有一定的提高