

并行分布式计算期末 Project

计算机科学技术学院 周训哲 20307110315

2023 年 5 月 31 日

多线程并行快速排序算法

本实验的实现过程主要是使用 C++ 版本的 openMP 编程实现并行快速排序算法。在测试加速比的过程当中使用了 1000, 5000, 10000, 100000, 1000000, 10000000, 100000000 的数据量, 发现随着数据量的增加, 并行计算的加速比越大, 符合预期。

一、模型介绍

openMP 介绍

OpenMP (Open Multi-Processing) 是一种并行编程模型, 用于在共享内存系统中编写多线程并行程序。它提供了一组指令和编译器指导, 可以将串行程序转换为并行程序, 以充分利用多核处理器和共享内存体系结构的性能。

OpenMP 的编程模型基于共享内存的概念, 其中多个线程可以并行执行任务。在 OpenMP 中, 程序员使用指令和编译器指导来标识可并行化的代码段, 并指定线程之间的同步和数据共享方式。这样, 程序员可以将任务分解成多个并行的线程, 每个线程负责处理任务的一个部分。

快速排序介绍

快速排序 (Quick Sort) 是一种常用的排序算法, 它基于分治 (Divide and Conquer) 的策略来进行排序。快速排序的核心思想是通过选择一个基

准元素，将待排序的序列分割成两个子序列，使得左侧子序列中的所有元素都小于等于基准元素，右侧子序列中的所有元素都大于等于基准元素，然后对两个子序列递归地应用快速排序算法，直到序列变为有序。

快速排序是一种天然适合并行化的排序算法，因为它的分治策略可以很容易地将排序任务分解成多个子任务，每个子任务独立地对子序列进行排序。这使得快速排序可以通过并行计算来加速排序过程，提高排序的效率。

在并行快速排序中，可以使用多线程或并行计算模型来同时处理多个子任务。每个线程或计算单元可以负责处理一个子序列，使用快速排序算法进行排序。然后，将子序列合并成最终的有序序列。

常见的并行快速排序算法的步骤：

1. 并行划分：选择一个基准元素，并使用并行计算或多线程来划分序列，使得每个线程或计算单元独立地处理子序列的一部分。每个线程或计算单元将小于等于基准元素的元素放在左侧，大于等于基准元素的元素放在右侧。

2. 并行排序：对左侧和右侧的子序列分别应用并行快速排序算法，每个线程或计算单元递归地对子序列进行排序。

3. 合并结果：等待所有线程或计算单元完成排序后，将各个子序列按照顺序合并，得到最终的有序序列。

二、模型实现

代码主要使用 C++ 进行实现，调用 `<omp.h>` 中的库函数，使用 `#pragma omp` 语句实现并行计算。

在实现 openMP 完成快速排序的过程中，主要将代码划分成 `quickSort` 和 `main` 两个部分，`quickSort` 定义了串行和并行算法的计算过程，主函数则调用算法对串行和并行算法的加速比进行计算。

同时为了实现生成固定长度的数据集以及一个串行算法用于测算对应的加速比，

三、测试结果

测试数据量	1000	5000	10000	100000
并行处理时间	0.000206 s	0.000632 s	0.001312 s	0.014380 s
串行处理时间	0.000256 s	0.000856 s	0.001788 s	0.021190 s
加速比	1.241881	1.355978	1.363345	1.473619

测试数据量	1000000	10000000	100000000
并行处理时间	0.150549 s	1.477709 s	9.344500 s
串行处理时间	0.241012 s	2.706674 s	18.478509 s
加速比	1.600890	1.831669	1.977474

```
-----  
测试数据量 : 1000  
并行处理时间 : 0.000206 s  
串行处理时间 : 0.000256 s  
加速比 : 1.241881  
-----  
  
-----  
测试数据量 : 5000  
并行处理时间 : 0.000632 s  
串行处理时间 : 0.000856 s  
加速比 : 1.355978  
-----  
  
-----  
测试数据量 : 10000  
并行处理时间 : 0.001312 s  
串行处理时间 : 0.001788 s  
加速比 : 1.363345  
-----  
  
-----  
测试数据量 : 100000  
并行处理时间 : 0.014380 s  
串行处理时间 : 0.021190 s  
加速比 : 1.473619  
-----  
  
-----  
测试数据量 : 1000000  
并行处理时间 : 0.150549 s  
串行处理时间 : 0.241012 s  
加速比 : 1.600890  
-----  
  
-----  
测试数据量 : 10000000  
并行处理时间 : 1.477709 s  
串行处理时间 : 2.706674 s  
加速比 : 1.831669  
-----  
  
-----  
测试数据量 : 100000000  
并行处理时间 : 9.344500 s  
串行处理时间 : 18.478509 s  
加速比 : 1.977474  
-----
```

图 1: openMP 快排加速比

MapReduce 实现 wordcount

本实验有两种实现方式，其中一种是使用 openMP 并行框架实现（使用 C++ 实现）；另外一种是使用 hadoop 实现（使用 python 实现）。在测试加速比过程中为了对比 hadoop 的加速的提升，使用 python 计算了 1K、1M、10M、100M 数据量下的加速比。

一、模型介绍

hadoop 介绍

Hadoop 是一个开源的分布式计算框架，旨在处理大规模数据集的存储和处理。它提供了可靠性、可扩展性和容错性，使得它成为处理大规模数据的理想选择。

Hadoop 的核心组件包括以下几个部分：

1.Hadoop 分布式文件系统（Hadoop Distributed File System, HDFS）：HDFS 是 Hadoop 的分布式文件系统，用于存储大规模数据集。它具有高容错性和可靠性，通过将数据分布在集群中的多个节点上实现数据冗余和容错。

2.Hadoop YARN（Yet Another Resource Negotiator）：YARN 是 Hadoop 的资源管理器，负责集群资源的管理和作业调度。它允许用户在 Hadoop 集群上运行各种计算框架，如 MapReduce、Spark 和 Flink 等。

3.MapReduce：MapReduce 是 Hadoop 的计算模型和编程范式。它将大规模数据集分割为小的数据块，并将其分发到集群中的多个计算节点上进行并行处理。MapReduce 提供了数据处理的分布式框架，包括数据划分、并行计算、数据排序和结果汇总等功能。

MapReduce 介绍

MapReduce 是一种分布式计算模型和编程范式，最初由 Google 提出，并成为 Hadoop 中的核心组件之一。它旨在解决大规模数据集的并行处理问题。

MapReduce 模型基于两个主要操作：Map 操作和 Reduce 操作。下面是对 MapReduce 模型的简要说明：

1.Map 操作：Map 操作是将输入数据集中的每个元素映射到一组键-值对（Key-Value Pair）。Map 操作通常会并行处理数据集的各个部分，将输入数据划分为多个片段，然后每个片段由一个 Map 任务处理。Map 任务对每个输入元素应用一组操作，生成键-值对作为中间结果。

2.Shuffle 和排序：在 Map 操作之后，框架会自动执行 Shuffle 和排序操作。Shuffle 的目的是将 Map 任务输出的键-值对重新分配到不同的 Reduce 任务上。在 Shuffle 过程中，相同键的键-值对会被分组并发送到同一个 Reduce 任务进行处理。此外，Shuffle 还会对键进行排序，以便在 Reduce 阶段进行更高效的处理。

3.Reduce 操作：Reduce 操作接收 Shuffle 和排序阶段的输出作为输入，并根据键对值进行聚合、汇总或其他计算。Reduce 任务会针对每个唯一的键处理一组值，生成最终的输出结果。Reduce 操作通常也是并行处理的，多个 Reduce 任务可以同时处理不同的键集。

二、模型实现

代码在 openMP 过程中主要使用 C++ 进行实现，调用 `<omp.h>` 中的库函数，使用 `#pragma omp` 语句实现并行计算。

在使用 hadoop 过程中主要使用 Python 进行实现，分别编写 mapper 和 reducer 进行 map 和 reduce 的操作。

在实现 openMP 完成快速排序的过程中，主要将代码划分成 quickSort 和 main 两个部分，quickSort 定义了串行和并行算法的计算过程，主函数则调用算法对串行和并行算法的加速比进行计算。

三、测试结果

最终测试时采用单个线程与 8 个线程的 mapreduce 算法进行比较计算加速比。

可以看到随着数据量的提升，加速比也有十分大的提升。

测试数据量	1000	1M	10M	100M
串行处理时间	0.0147531 s	12.2467 s	170.093 s	3326.44 s
并行处理时间	0.0145879 s	8.33072 s	80.9971 s	786.0211 s
加速比	1.0113	1.4701	2.0981	4.2319

python 程序中测试使用 Hadoop 的结果对于长度 1M 的数据需要运行 9s，而使用 python 中的 map 函数则只需要 0.3s

The screenshot displays the Hadoop web interface for a specific application. The top navigation bar includes links for Cluster, About, Nodes, Node Labels, Applications, NEW, NEW_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, and Scheduler. The main content area is titled 'Application application_1684910638268_0002' and shows the following details:

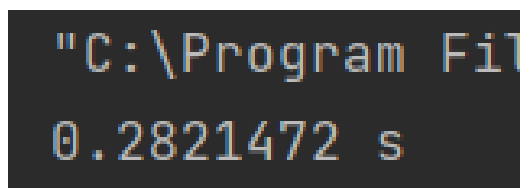
- User:** zhouxunzhe
- Name:** streamjob4561459166066681396.jar
- Application Type:** MAPREDUCE
- Application Tags:**
- YarnApplicationState:** FINISHED
- FinalStatus Reported by AM:** SUCCEEDED
- Started:** 星期三 五月 24 14:45:47 +0800 2023
- Elapsed:** 19sec
- Tracking URL:** History
- Diagnostics:**

Below the application details, the 'Application Metrics' section shows:

- Total Resource Preempted:** <memory:0, vCores:0>
- Total Number of Non-AM Containers Preempted:** 0
- Total Number of AM Containers Preempted:** 0
- Resource Preempted from Current Attempt:** <memory:0, vCores:0>
- Number of Non-AM Containers Preempted from Current Attempt:** 0
- Aggregate Resource Allocation:** 81457 MB-seconds, 51 vcore-seconds

At the bottom, there is a table with columns for Attempt ID, Started, Node, and Logs. The table shows one entry with Attempt ID 'appattempt_1684910638268_0002_000001', Started at 'Wed May 24 14:45:47 +0800 2023', and Node 'http://laptop:8042'. The table is showing 1 to 1 of 1 entries.

图 2: 使用 hadoop 进行 mapreduce 计算



```
"C:\Program Fil  
0.2821472 s
```

图 3: 使用 python 中的 map 库进行 wordcount

实验总结

在本次实验中，我第一次接触到并行计算的代码实现。在本节课程以及本次课程 project 中，我学会了很多并行编程的技巧和掌握了并行编程的能力。除此以外，我觉得最为关键的就是我自己搭建了 Hadoop 的计算环境。该项工作为我之后的大数据处理奠定了一定的基础。通过本次 pj，我更加深刻理解了 openMP 和 MapReduce 的编程原理以及并行计算对于大数据计算的重要性。

非常感谢老师以及助教对本次项目的设计以及问题的解答。