南開大學
Nankai University

计算机学院

并行程序设计实验报告

期末开题报告

马浩祎

学号：2213559

专业：计算机科学与技术

2024 年 4 月 7 日

目录

1 摘要 **2**

2 选题 **2**

2.1 问题描述 2

2.1.1 倒排索引 2

2.1.2 倒排索引求交 2

3 背景知识 **3**

3.1 经典串行算法 3

3.2 多 CPU 并行算法 4

3.3 GPU 并行算法 5

3.4 简要总结 7

4 期末研究方案 **8**

4.1 串行研究 8

4.2 并行研究 8

4.2.1 串行算法并行化 8

4.2.2 SIMD 8

4.2.3 Pthread/OpenMP 9

4.2.4 MPI 9

4.2.5 GPU 10

4.3 性能指标 10

5 总结 **11**

1 摘要

Abstract

本次实验报告主要分为两个部分：在选定倒排索引求交问题后对前人研究工作的梳理总结，涉及多方面的研究方向：串行，多 CPU 并行，GPU 并行，有序无序列表，Bloom 等工具的使用优化；规划自己期末研究的方法流程，结合平时四次并行架构编程 SIMD、Pthread/OpenMP、MPI、GPU 进行延伸，探索 GPU 批次集合求交算法，并对可能涉及的 GPU 编码解压缩处理进行方案探索，同时选定出针对本问题的性能指标。希望通过开题的起点，为之后的研究长路打下坚实的基础。

关键词：倒排索引，集合求交，并行计算

2 选题

本学期我的期末研究选题是倒排索引求交问题。

2.1 问题描述

2.1.1 倒排索引

倒排索引，又名反向索引、置入文档等，多使用在全文搜索下，是一种通过映射来表示某个单词在一个文档或者一组文档中的存储位置的索引方法。在各种文档检索系统中，它是最常用的数据结构之一。

那么既然称其为倒排索引，反向索引，对应的，我们需要额外理解一下什么是正向索引。所谓正向索引，就是当用户发起查询时，搜索引擎会扫描索引库中的所有文档，找出所有包含关键词的文档，这样依次从文档中去查找是否含有关键词的方法。那很显然，这个搜索量是巨大的，算法效率低。

再来清楚的看一下两者的索引结构：

- 正向：文档-> 单词、单词、单词……
- 反向：单词-> 文档、文档、文档……

我们假设有一个数据集，包含 n 个网页或文档，现在我们想将其整理成一个可索引的数据集。我们当然可以按照正向索引，逐一遍历网页，为其内部的单词建立索引，但是正如前文所说，这样效率低下。具体地，利用倒排索引，我们可以为数据集中的每篇文档选取一个文档编号，使其范围在 [1,n] 中。其中的每一篇文档，都可以看做是一组词的序列。则对于文档中出现的任意一个词，都会有一个对应的文档序列集合，该集合通常按文档编号升序排列为一个升序列表，即称为倒排列表（Posting List）。所有词项的倒排列表组合起来就构成了整个数据集的倒排索引。这和我们的索引结构是相符的。

2.1.2 倒排索引求交

有了倒排索引后，搜索引擎在查询处理过程中，需要对这些索引集合进行求交操作，这个也是运算时间占比非常大的部分。假定用户提交一个 k 个词的查询，查询词分别是 $t_1, t_2, \ldots, t_k$，这 k 个关键词对应的倒排列表为 $\ell_{t_1}, \ell_{t_2}, \ldots, \ell_{t_k}$ 求交算法返回 $\cap_{1 \le i \le k} l(t_i)$。

首先求交会按照倒排列表的长度对列表进行升序排序，使得：

$$|l(t_1)| \le |l(t_2)| \le \cdots \le |l(t_k)|$$

然后求交操作返回 k 个倒排列表的公共元素 $\cap_{1 \le i \le k} l(t_i)$。

举个例子：比如查询 “2024 Spring ”，搜索引擎会在索引中找到 “2024”，“Spring” 对应的倒排列表，假定为:

$$l(2024) = (13, 14, 15, 16, 17)$$

$$l(Spring) = (4, 8, 11, 13, 14, 16)$$

这是排好序的倒排列表，下面求交获得：

$$l(2024) \cap l(Spring) = (13, 14, 16)$$

但是这里只给出了输入和输出。如何进行求交操作并未提出，而如何设计出高效的并行求交算法正是我研究的问题关键。

3 背景知识

其实这个问题如果在没有进行索引压缩时，它的本质就是多个有序数组的求交。

3.1 经典串行算法

1970 年初，Hwang 和 Lin 研究了两个有序数组的求交串行算法 [3]。根据他们的介绍，可以得出经典串行算法之一的伪代码：

Algorithm 1 求交串行算法

Input: 两个有序数组 A 和 B

Output: $A \cap B$

```
1:  function FUNCTION(A[], B[])
2:      i ← 0
3:      j ← 0
4:      while i < |A| andj < |B| do
5:          if A[i] == B[j] then
6:              A[i] 添加到 A ∩ B 中
7:              i ← i + 1
8:              j ← j + 1
9:          else
10:             if A[i] < B[j] then
11:                 i ← i + 1
12:             else
13:                 j ← j + 1
14:             end if
15:         end if
16:     end while
17: end function
```

可以看到其时间复杂度为 $O(m+n)$，其中 m 和 n 分别为两个有序数组的元素个数。随后 Demaine 将他们的工作成果推广到多个有序数组上，还提出了高效的自适应算法。[1] 但是我们主要针对并行算法进行研究，这里不多赘述。

3.2 多 CPU 并行算法

2009 年 Tatikonda 提出了针对多 CPU 的两级并行算法 [5]，提到了查询内和查询间两个层面的并行处理，如图 2.1（采自作者的论文）。这里简要说明一下：

- 查询内并行化：在单个查询中对索引列表求交进行并行计算。在该策略下，对于一个查询，倒排索引中的多个 list 被划分为多个子任务，每个子任务在不同的核心或线程中并行处理，同时持有一个被执行的交集。这可以利用多核处理器的并行计算能力，提高交集计算的效率。
- 查询间并行化：在多个查询之间对索引列表求交进行并行计算。在该策略下，不同的查询被分配给不同的核心或线程进行独立的计算。这样可以同时处理多个查询的交集计算，进一步提高性能。主要采用基于合并的技术和跳跃的修剪策略等方法。

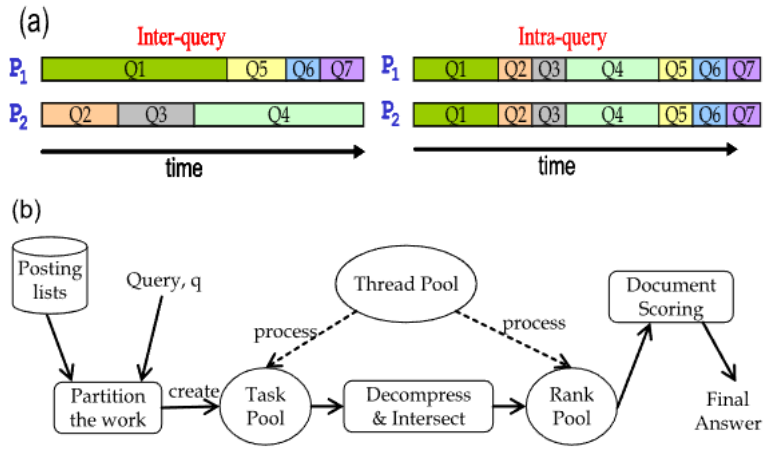


图 3.1: Caption

同年，Tsirogiannis 提出了动态探测算法 [6]，该策略可以在运行时利用以前的探测信息动态决定列表上的探测顺序。该信息被用作在高速缓存驻留的微索引。这是针对排序列表的，他同时也提出了对于未排序列表的算法策略：一种新的基于哈希的算法，我们知道哈希可以有效避免排序的开销。

上面提到的两个算法均有着负载均衡的特点，同时利用了 CPU 的多级缓存，使得降低了访问延迟，而且对于未排序列表的算法策略，有着极高的性能。由于作者提到他是在 8 核上进行的实验，故可以认为这也是并行策略的体现。

由图 2.2（作者论文中的指标图）可以看出，Tsirogiannis 的两个算法都有着极高的性能（DP 和 QB），这里 LM 是普通的线性求交算法，SA 是 2.1 节提到的 Demaine 的自适应算法 Small Adaptive，PP 是 2007 年提出的 Probabilistic Probes 算法。可以看到，本文所回顾的学者们已经有了前后的联系。

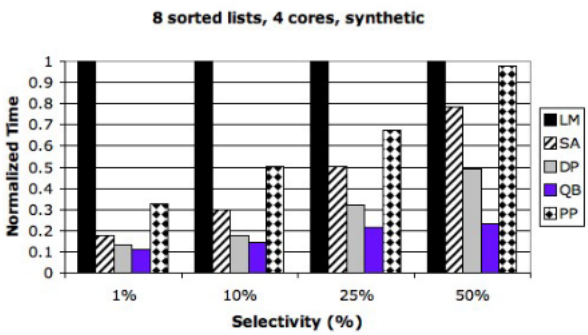
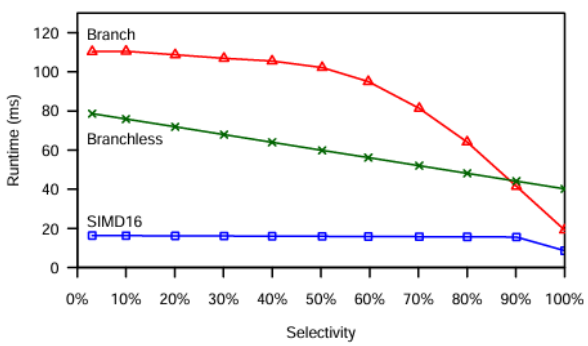
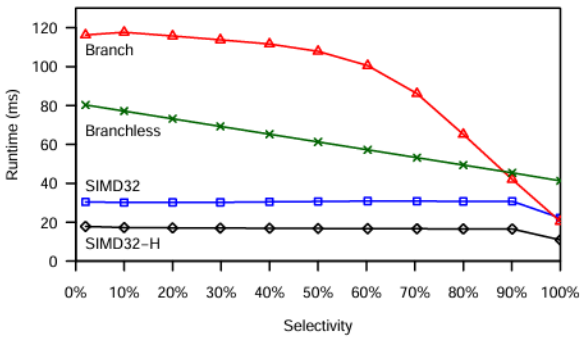


图 3.2: Caption

2011 年，Schlegel 等人利用 SSE 指令集的特点和高效 SIMD 指令来优化指令数，这是一种并行算法，而且他们针对不同的整数数据类型，提出了不同的排序集交集算法，注意这里是排序的列表。他们还采用了定制数据布局的方法来优化性能。[4] 针对不同的数据类型，这里给出作者论文中的指标图，如图 2.3，我们可以看到他们的算法的性能良好。



(a) SIMD-16



(b) SIMD-32

图 3.3: SIMD 指令优化的效率图

3.3 GPU 并行算法

该研究领域中除了多 CPU 并行算法，也有学者研究过利用 GPU 的归并计算，比如 Ding 在 2008 年提出的利用其进行查询处理的架构。[2] 他们的原型是基于 CUDA 的，且在如图 2.4（采自作者的论文）的新架构上进行实验，最终能获得 2 倍以上的加速比。但我注意到他们的查询需要编码解压缩等等步骤，且 GPU 会参与这些步骤，故我们不再深究其中原因，只作为背景参考。

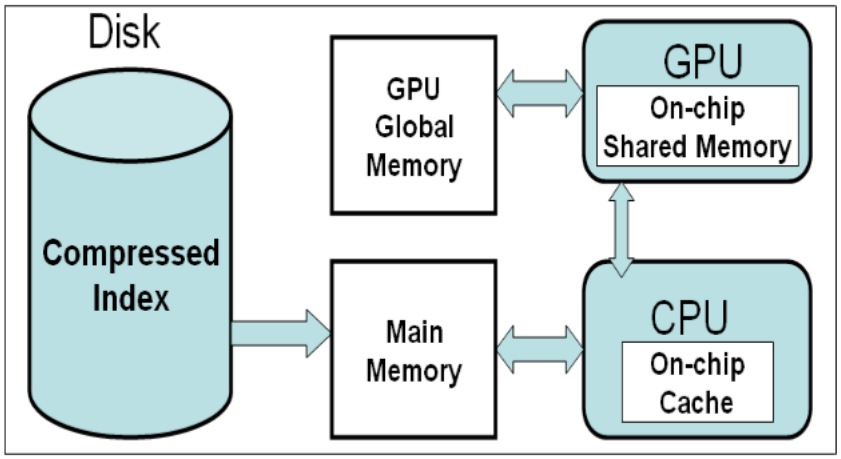


图 3.4: 基于 GPU 的系统架构

2009 年，南开大学的南开百度联合实验室提出了针对倒排索引求交的 GPU 算法 [7]。该算法将查询分批提供给 GPU，如图 2.5（采自作者的论文），因此可以占用全部时间 GPU 处理器核心的优势，即使问题规模很小。这一点不同于刚才提到的 Ding 提出的算法，后者的算法在小规模倒排索引的效果不甚理想。而前者还提出了一种输入预处理方法来解决负载不平衡问题。他们的实验结果表明，分批算法要比最快的 CPU 算法和普通 GPU 算法快很多。如图 2.6（采自作者的论文），我们可以看到不论查询数量多少，算法的效率随着批次的上涨会有所提升。

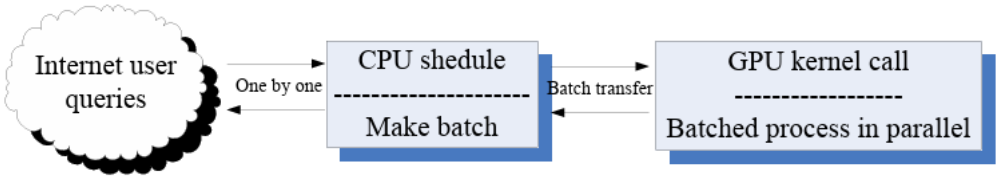


图 3.5: 分批算法结构

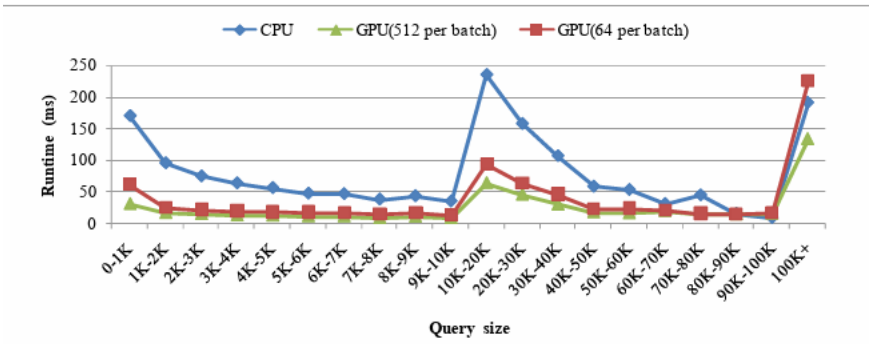


图 3.6: 分批算法指标图

最后，我们来看一下 2011 年南开大学的南开百度联合实验室提出的另外一个方法 [8]。他们利用 Bloom Filter 来进行 GPU 集合求交，Bloom Filter 具有高效的插入和查询操作，并且占用的内存空间相对较小。然而，由于使用了哈希函数和位数组，Bloom Filter 存在一定的误判率，即有可能存在“假阳性”判断。团队为每个列表生成 Bloom 过滤器，最终取得了良好的加速效果，如图 2.7（采自作

者的论文)，BS 指二进制查找，BF 指 Bloom 过滤器优化，GOV 和 Baidu 指的是两类数据集，指标是吞吐量，显然 BF 的吞吐量要高于 BS。但是正如 Bloom 过滤器的误判弊端，该算法存在不能保证返回所有公共 docID 的问题。不过，值得说明的是，该算法也针对有序列表。

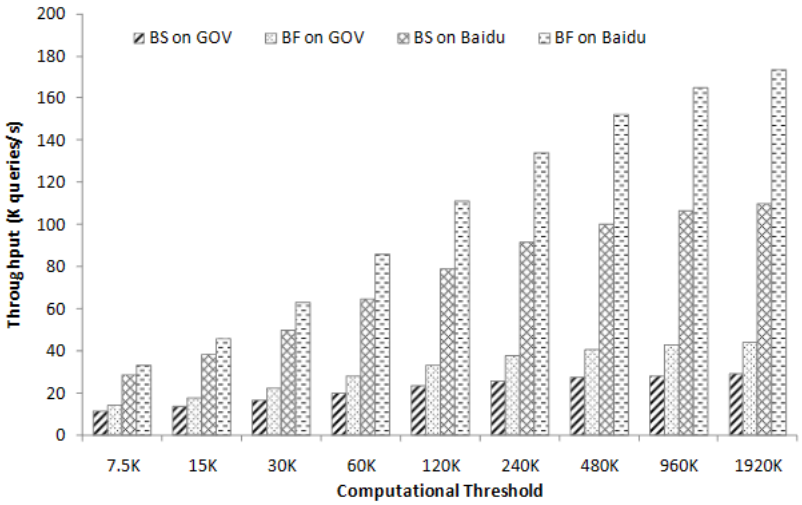


图 3.7: Bloom 优化效果图

3.4 简要总结

在本节，我们回顾了该领域前人所作的相关工作，尤其是并行求解的方面。

首先，我们可以看到，此问题最初的串行求解算法易于理解，也有着串行算法中较好的时间复杂度。针对多个有序数组的研究也正是我将要去探索的 baseline 内容。

然后，在并行算法中，我们看到了多 CPU 算法和 GPU 参与的算法两类领域的研究。

- 多 CPU：可以看到研究者们已经将 CPU 挖掘地极其充分
  - 研究者们在查询内和查询间进行两层的并行分工，充分利用 CPU 多核心线程的优势
  - 研究者们利用 CPU 高速缓存的特点，开发出动态探测，哈希等算法。同时后者也是涉及未排序列表的一个范例
  - 研究者们还利用 CPU 架构中 SIMD 指令集的特点来优化指令数，定制布局，以达到优化性能
- GPU：在新兴 CUDA 下，研究者们提出了丰富的 GPU 参与的查询架构
  - 研究者们提出 GPU 参与解压缩，跳转等步骤的复杂并行算法，但是针对规模小的查询性能较弱
  - 南开大学的研究者们针对倒排索引求交提出由 CPU 分批送给 GPU 进行处理的算法，能适应任意的查询规模，同时也通过输入预处理解决负载不均衡的问题。研究者们还借助新兴工具进行求交的优化，取得了良好的加速比，但是由于工具副作用，存在极少量虚假判断的问题

总的来说，该问题在串行、并行，列表的有序、无序和其他工具的使用与否等方面都有充分的研究成果，这拓展了我的思路，为我未来的探究打下了坚实的基础。

4 期末研究方案

4.1 串行研究

实际上的多链表求交主要有两种方式：按表求交（list-wise intersection）和按元素求交（element-wise-intersection），我将实现这两种思路的串行算法，同时作为并行性能的 baseline。

按表求交策略是先使用两个表进行求交，得到中间结果再和第三条表求交，依次类推直到求交结束。这样求交的好处是，每一轮求交之后的结果都将变少，因此在接下来的求交中，计算量也将更少。按元素求交算法会整体的处理所有的升序列表，每次得到全部倒排表中的一个交集元素。这类算法可以较好的应用算分策略和提前停止技术。下面简要介绍其一串行算法的伪代码：

Algorithm 2 表求交串行算法

Input: $\boldsymbol{l}(\boldsymbol{t}_1), \boldsymbol{l}(\boldsymbol{t}_2), \ldots, \boldsymbol{l}(\boldsymbol{t}_k)$, Sorted by $|\boldsymbol{l}(\boldsymbol{t}_1)| \leq |\boldsymbol{l}(\boldsymbol{t}_2)| \leq \cdots \leq |\boldsymbol{l}(\boldsymbol{t}_k)|$

Output: $\cap_{1 \leq i \leq k} l(t_i)$

```
function FUNCTION(l(t_1), l(t_2), ..., l(t_k))
    S ← l(t_1)
    for i = 2 → k do
        for each element e ∈ S do
            found = find(e, l_i)
            if found = false then
                Delete e from S
            end if
        end for
    end for
end function
```

4.2 并行研究

由于平时编程作业会以 4 种形式布置，这也对应着 4 种并行架构。故我选择结合期末选题进行学习研究。

4.2.1 串行算法并行化

我准备对串行算法进行优化改进，参照以前体系结构编程中的有关思路：

- 观察数组访问形式，思考是否有改进方案使得增加 cache 命中率。
- 观察指令，思考是否可以利用超标量架构实现相邻指令的无依赖。比如说上面按表求交，注意到每个列表对于表 1 都是独立的，可以分多路求 k-1 个列表的交集。
- 观察算法，思考可否用更好的串行算法来减少比较次数，减少指令条数。

4.2.2 SIMD

SIMD 是单指令流多数据流模式，我可以采用基于位图存储的倒排链表表示：

将每条链表用一个位向量表示，其中每个 bit 对应一个 DocID，1 表示该链表包含此 Doc，0 表示不包含。从而将倒排索引中的每个链表转换为位图表示。可以有以下参考思路：

- 将两个位图进行位与运算，得到两个位图的交集。使用 SIMD 指令集（SSE、AVX 等）实现对位图的并行位与运算。利用 SIMD 指令一次处理多个位向量元素，提高并行效率。
- 针对稀疏情况，可以建立二级索引：将位向量分块，每个块用一个 bit 表示其全 0 还是非全 0，这些 bit 构成二级位向量。只有二级位向量位与为 1 时才进行底层位向量对应块的位与运算。这样可以减少位与运算的次数，提高效率。
- 还可采用 DocID 重排：对 DocID 进行重排，使得位图更加聚集，减少位向量中的零值数量。可使用类似于分块排序的方法，将相邻的 DocID 分配到相同的块中。重排后的位图更适合进行 SIMD 并行位与运算，提高性能。

4.2.3 Pthread/OpenMP

POSIX 线程（POSIX Threads，常被缩写为 pthreads）是 POSIX 的线程标准，定义了创建和操纵线程的一套 API。实现 POSIX 线程标准的库常被称作 pthreads，一般用于 Unix-like POSIX 系统，如 Linux、Solaris。但是 Microsoft Windows 上的实现也存在，例如直接使用 Windows API 实现的第三方库 pthreads-w32。这是多线程编程的常用工具，库函数架构如图 3.8。我准备在多个平台上进行多线程并行编程的观察和研究。这里由于不涉及 GPU，可以参照前人研究多 CPU 时的策略：

- 查询间并行，不同线程处理不同 Query，额外开销少，有利于高吞吐率，但不能优化响应延迟。
- 查询内并行，能优化大 Query 的响应延迟，但有额外开销，不利于高吞吐率。
- 尝试两级查询并行方法。进一步提高性能。

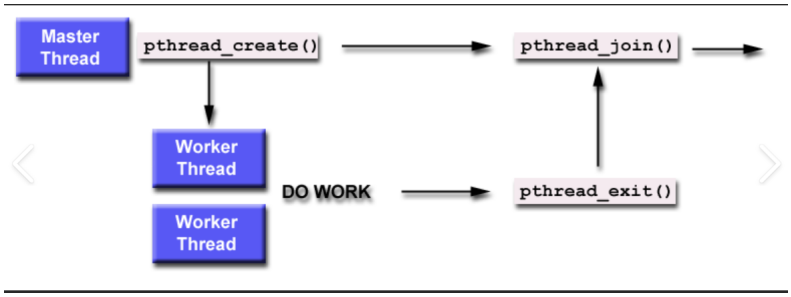


图 4.8: Pthread 函数架构

另一方面，OpenMP（Open Multi-Processing）是一套支持跨平台共享内存方式的多线程并发的编程 API，使用 C,C++ 和 Fortran 语言，可以在大多数的处理器体系和操作系统中运行，包括 Solaris, AIX, HP-UX, GNU/Linux, Mac OS X, 和 Microsoft Windows。包括一套编译器指令、库和一些能够影响运行行为的环境变量。因此我也打算评估工具之间的多线程并行效果差异。具体方法和 Pthread 差不多。

4.2.4 MPI

消息传递接口（Message Passing Interface，缩写 MPI）是一个并行计算的应用程序接口（API），常在超级计算机、电脑集群等非共享内存环境程序设计。这里需要考虑通信开销及其优化，为了证明其可行性，这里给出程序设计时可能的流程，如图 3.9（取自知乎）。设计时可以分为以下几步：

- 数据划分将输入数据划分为多个子集，每个子集分配给不同的线程。

- 确定进程之间的通信模式，例如点对点通信和集合通信，并比较差异。
- 使用 MPI 通信机制将本地结果发送到其他进程进行同步。测算响应时间和吞吐率。

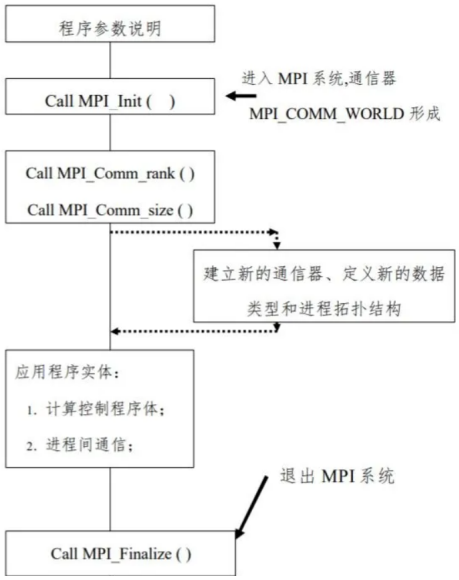


图 4.9: MPI 函数架构

4.2.5 GPU

该部分我想重点研究，也是期末研究中拟拓展的部分。基本研究的具体思想可以分为两部分，线程块间和线程块内。类似于多线程和 SIMD 编程，这里需要注意：

- 学习 CUDA 相关知识，利用该编程框架管理 GPU 的内存分配和数据传输
- 设计 GPU 上的并行求交算法，还可利用 GPU 的 SIMD 能力，处理多个数据元素
- 优化时还可以通过调整线程块大小，共享内存大小等参数来优化算法性能

对于拓展研究，根据敖博士的论文 [9]，他主要对 GPU 的并行求交进行了优化，我拟简单实现批次集合求交部分。其实我对信息时代很关键的倒排索引问题很感兴趣，我想参考敖博士的论文理解索引的编码和解压缩部分，有机会的话实现一下 GPU 的并行解压缩处理，并进行性能的对比。

4.3 性能指标

不管是串行算法还是并行算法，我需要对各种算法思路进行实现并收集数据，对比性能，给出分析，这里针对倒排索引求交问题，拟采用指标：

- 响应延迟，即一个 Query 处理所需时间
- 吞吐率，即每秒 Query 处理量
- 并串算法对比的加速比等指标的分析
- 如果涉及到编码解压缩等部分，还会有压缩比等指标分析

5 总结

本次报告为并行课程的期末开题报告，主要探讨了以下问题：

- 对倒排索引求交问题的核心概念理解。
- 对前人工作的阅读和概括，涉及多方面的研究方向：串行，多 CPU 并行，GPU 并行，有序无序列表，哈希等工具的使用优化。
- 对期末研究问题大概框架的确定以及详细内容方法的梳理。
- 对期末拓展部分进行展望，给出可能的探索方案。
- 对期末研究的性能指标给出明确的选择。

总之，这次报告帮我梳理了期末研究的步骤流程，可能后续会有所出入，但这确实给我的期末研究指明了方向。(未来有关期末研究的实验资料均在https://github.com/Mhy166/parallel-programming.git仓库中)

参考文献

[1] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, page 743–752, USA, 2000. Society for Industrial and Applied Mathematics.

[2] Shuai Ding, Jinru He, Hao Yan, and Torsten Suel. Using graphics processors for high-performance ir query processing. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, page 1213–1214, New York, NY, USA, 2008. Association for Computing Machinery.

[3] F. K. Hwang and S. Lin. A simple algorithm for merging two disjoint linearly ordered sets. *SIAM Journal on Computing*, 1(1):31–39, 1972.

[4] Benjamin Schlegel, Thomas Willhalm, and Wolfgang Lehner. Fast sorted-set intersection using simd instructions. In *ADMS@VLDB*, 2011.

[5] Shirish Tatikonda, Flavio Junqueira, B. Barla Cambazoglu, and Vassilis Plachouras. On efficient posting list intersection with multicore processors. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 738–739, New York, NY, USA, 2009. Association for Computing Machinery.

[6] Dimitris Tsirogiannis, Sudipto Guha, and Nick Koudas. Improving the performance of list intersection. *Proc. VLDB Endow.*, 2(1):838–849, aug 2009.

[7] Di Wu, Fan Zhang, Naiyong Ao, Fang Wang, Xiaoguang Liu, and Gang Wang. A batched gpu algorithm for set intersection. In *2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks*, pages 752–756, 2009.

[8] Fan Zhang, Di Wu, Naiyong Ao, Gang Wang, Xiaoguang Liu, and Jing Liu. Fast lists intersection with bloom filter using graphics processing units. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, page 825–826, New York, NY, USA, 2011. Association for Computing Machinery.

[9] 敖耐勇. 多核/众核平台下索引压缩及集合求交并行算法研究. PhD thesis, 南开大学, 2017.