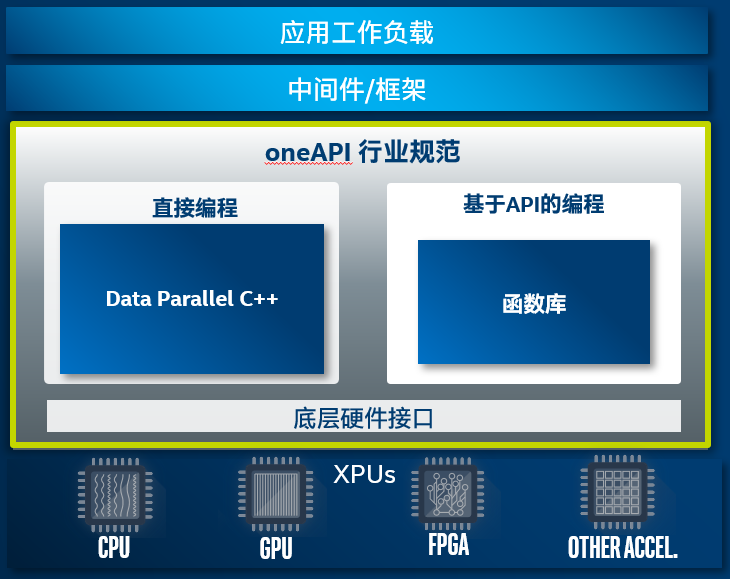
期末研究报告选题示例

注意，以下仅仅是可能的选题例子的简单介绍，并非完整的开题报告示例。

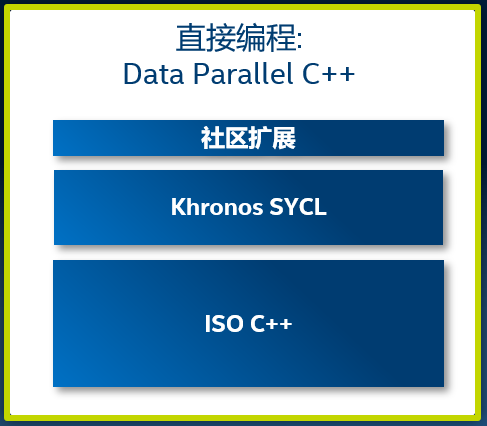
## 选题示例一——OneAPI并行开发工具研究（工具研究类）

### 1、OneAPI及其开发生态简介

OneAPI是Intel推出的跨架构（多核CPU、众核GPU、FPGA、…）统一编程模型，旨在简化多种架构下并行程序的开发流程。针对并行化表达对语言和函数库进行统一和简化，同时又不牺牲性能。基于行业标准和开放式规范，能够与现有 HPC 编程模型互操作。

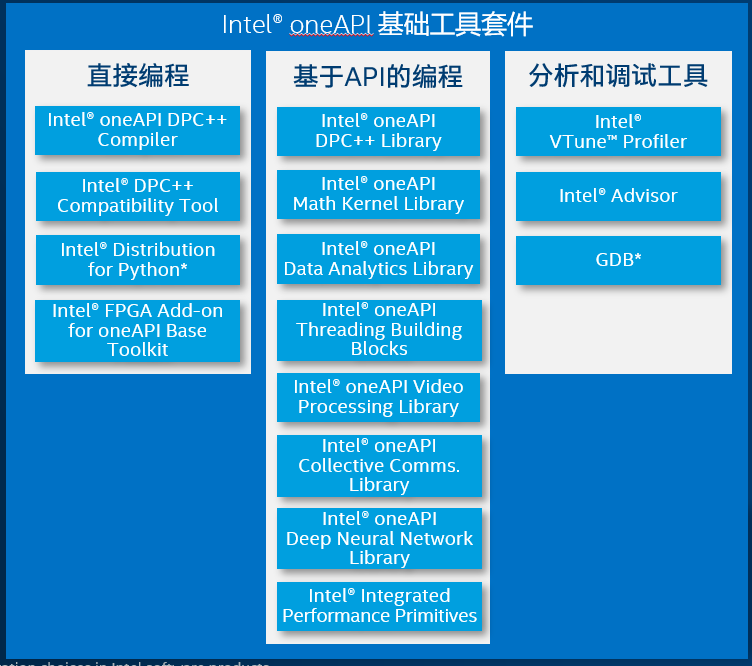


其主要组件是跨架构并行编程语言DPC++（Data parallel C++）。它基于ISO C++ 和Khronos SYCL，提供跨CPU和加速器的卓越的并行编程效率和性能，依托社区项目推动语言的增强、不断演进。



形成了完成的开发生态，包括核心工具套件和库——Data Parallel C++编译器、库和分析工具，支持跨架构（CPU、GPU、FPGA）开发高性能应用：

* DPC++语言实现代码在不同架构间的复用，而DPC++编译器实现针对不同架构的目标代码调优，在实现统一编程便捷性的同时又不牺牲性能。
* DPC++ Compatibility tool可帮助迁移用CUDA编写的现有代码到DPC++程序，可实现80-90%代码的自动迁移，提供行内注释帮助程序员完成剩余迁移。
* Python分发版包括OneAPI加速的 scikit-learn、NumPy和SciPy库。
* 优化的性能库，支持线程化、数学、数据分析、深度学习和视频/图像/信号处理：
  + DPC++库：C++17算法的优化、基于并行STL、Boost等著名的并行库、集成兼容工具。
  + 视频处理库：加速媒体、视频处理，跨架构的简单但强大的API。
  + 深度神经网络库：跨架构、跨操作系统支持，开源。
  + 组通信库：利用组通信优化分布式机器学习，优化组通信实现。
* 分析调试工具：
  + Vtune Profiler：分析DPC++程序，针对不同架构进行调优，OpenMP卸载调优，多方面的性能剖析（线程、内存、cache、…），多种高级语言支持。
  + Advisor：估计卸载（到GPU等加速器）性能，优化访存和计算性能，向量化建议，多线程建议，流图分析。
  + GDB：对DPC++程序调试的支持。



### 2、研究思路参考

1. 作为工具研究载体的实际问题载体。可选择编程实验默认题目高斯消去法或其他类似问题——具有一定复杂度，“经得起”整套工具套件对其进行多方面研究。
2. **对比研究**。针对OneAPI工具套件“宣称”的特性，与传统工具进行对比研究。
   1. 基本编程方面，在不同并行架构下（SIMD、多核CPU、GPU），用专用编程工具/语言（SSE/AVX、Pthread、OpenMP、CUDA）对待求解问题进行并行求解，与用DPC++并行求解进行编程工作量、目标程序性能等方面的对比。
   2. 代码编写辅助方面，利用DPC++ Compatibility tool对CUDA程序进行到DPC++程序的迁移，对比CUDA程序、DPC++程序和采用Compatibility tool迁移的DPC++程序三者进行对比。类似的，可研究利用Advisor进行程序优化对性能的影响。
   3. 辅助库方面，若作为载体的问题是一般问题，可研究利用DPC++库来编写程序与不借助库在编程工作量、目标程序性能等方面的对比；如果是一些特殊问题可研究对应的库，如研究的是深度神经网络相关问题，可深度神经网络库进行类似研究。
   4. 分析调试工具方面，利用Vtune等工具进行DPC++程序的深层剖析，寻找性能瓶颈，进行相应优化，对优化后程序同样进行剖析，来分析优化思路是否奏效等。
3. 撰写研究报告。
4. 可在Intel的DevCloud平台完成研究工作，OneAPI工具套件的相关文档也可在其上找到，DevCloud的注册、使用详见更新的《实验教学指导书-1》。

## 选题示例二——华为鲲鹏计算加速库研究（工具研究类）

### 1、华为鲲鹏计算加速库简介

将应用迁移到鲲鹏计算平台后，可以选用华为优化后的计算加速库，为迁移后的软件提供性能保证，当前完成性能优化的加速库包括glibc、HMPP、Hyperscan、AVX2Neon、ZSTD、Snappy、gzip、x265。

glibc库主要对内存、字符串、锁等接口基于鲲鹏处理器微架构特点进行了加速优化，优化后代码通过鲲鹏开源社区公开发布，客户可以根据自己业务迁移需要下载使用。

HMPP主要提供基础的音视频信号高级函数接口，基于鲲鹏处理器微架构进行了性能优化和功能补齐，以降低客户高性能音视频应用迁移到鲲鹏平台的改造工作量。

Hyperscan加速库使用aarch64的intrinsics函数进行加速优化。

AVX2Neon加速库完成常用的265个AVX&SSE函数的接口适配和性能优化。

ZSTD&Snappy加速库通过对算法实现做优化，提升压缩解压性能。

gzip加速库鲲鹏指令优化关键热点，实现压缩解压的性能优化。

x265使用鲲鹏指令重构视频转码热点函数，实现转码性能的提升。

### 2、研究思路参考

1. 对每种加速库，选取适合的问题，采用普通实现方法进行求解（包括并行化），同时采用加速库实现优化求解，进行实验比较两种实现方法的性能。
2. 华为鲲鹏计算加速库的详细内容参考《华为鲲鹏计算加速库——技术白皮书》。
3. 选择本问题的同学了解鲲鹏计算加速库的基本情况、掌握了其安装部署方法后，在我们的实验平台统一安装部署，在其上进行实验。

## 选题示例三——倒排索引求交（算法研究类）

### 1、问题描述

倒排索引（inverted index），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](http://zh.wikipedia.org/w/index.php?title=%E5%AD%98%E5%82%A8%E4%BD%8D%E7%BD%AE&action=edit&redlink=1)的索引方法。在各种[文档检索系统](http://zh.wikipedia.org/w/index.php?title=%E6%96%87%E6%A1%A3%E6%A3%80%E7%B4%A2%E7%B3%BB%E7%BB%9F&action=edit&redlink=1)中，它是最常用的[数据结构](http://zh.wikipedia.org/wiki/%E6%95%B0%E6%8D%AE%E7%BB%93%E6%9E%84)之一。

对于一个有 个网页或文档（Document）的数据集，若想将其整理成一个可索引的数据集，则可以为数据集中的每篇文档选取一个文档编号（DocID），使其范围在中。其中的每一篇文档，都可以看做是一组词（Term）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。

倒排列表求交（List Intersection）也称表求交或者集合求交，当用户提交了一个 个词的查询，查询词分别是，表求交算法返回。

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：

例如查询“2014 NBA Final”，搜索引擎首先在索引中找到“2014”，“NBA”，“Final”对应的倒排列表，并按照列表长度进行排序：

求交操作返回三个倒排列表的公共元素，即：

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）。

按表求交基本思想是：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。具体算法描述见算法1：

|  |
| --- |
| Algorithm 1：Set versus Set 算法(SvS) |
| **Input:** , Sorted by  **Output:** Intersection Results |
| 1:  2: **for** to *k* **do**  3: **for** each element **do**  4:  5: **if** **then**  6: Delete *e* from *S*  7: **end if**  8: **end for**  9: **end for** |
| 10: **return** *S* |

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。这类算法通常在DAAT（Document at a time）查询下使用，以Adaptive算法为主，可以较好的应用算分策略和提前停止技术。

|  |
| --- |
| Algorithm 2：Adaptive 算法 (Adp) |
| **Input:**, Sorted by  **Output:** Intersection Results |
| 1:  2: **while** No empty list **do** |
| 3: Reorder the lists by increasing number of undetected elements.  4:  5:  6: **repeat**  7:  8:  9: **until** or  10: **if**  and **then**  11:  12: **end if**  13: **end while**  14: **return** *S* |

DAAT的基本思路就是在各个列表中寻找当前文档，当在所有列表中寻找完某个文档之后，每条链表的剩余的未扫描文档数量也不相同，但只要其中一条链表走到尽头，则本次求交结束。因此，每次都从剩余文档数量最短的链表进行扫描能够加速这个过程。而Adp算法的基本思路就是每次都从剩余文档数量最少的链表开始扫描，这样能够尽量缩短链表的扫描过程。

### 2、数据集描述

给定的数据集是一个截取自GOV2数据集的子集，格式如下：

1. ExpIndex是二进制倒排索引文件，所有数据均为四字节无符号整数（小端）。格式为：

[数组1]长度，[数组1]，[数组2]长度，[数组2]....

1. ExpQuery是文本文件，文件内每一行为一条查询记录；行中的每个数字对应索引文件的数组下标（term编号）。

注：ExpQuery文件在Windows系统下请使用写字板等工具打开。

百度盘链接: https://pan.baidu.com/s/1WiFWEplE7KaNaJpZzgOJgA 提取码: xqv1 复制这段内容后打开百度网盘手机App，操作更方便哦

鲲鹏服务器目录：/home/sTest/data/search\_engine

### 3、研究思路参考

实现list-wise和element-wise两种思路的串行算法，作为baseline，然后可从多个角度进行研究：

1. 对串行算法是否可以进行改进？如减少cache未命中次数、减少比较次数、实现指令级并行（相邻指令无依赖）以利用超标量架构等等。
2. 利用学习过的并行编程工具，设计实现并行求交算法：
   1. SIMD：升序列表求交操作本身不是标准的单指令流多数据流模式，很难直接进行向量化。因此，除了对倒排链表的存储格式及其上的串行求交算法直接进行SIMD并行化之外，还可考虑其他更利于向量化的存储方式和求交算法。例如，可考虑位图存储方式——每条链表用一个位向量表示，每个bit对应一个DocID，某位为1表示该链表包含此Doc、为0表示不包含。从而求交运算就变为两个位向量的位与运算。其优点是非常适合SIMD并行化（后续多线程、MPI、GPU等并行化也都很适合）、并行效率很高。缺点是倒排链表通常是比较稀疏的——表示为位向量的话绝大多数bit均为0、极少数为1，这就造成存储空间和串行计算时间可能都远不如倒排链表方式，即使并行效率很高可能最终性能也不如倒排链表。可考虑一些优化策略，例如建立二级索引——将位向量分块，每个块用一个bit表示其全0（0）还是非全0（1），这些bit构成二级位向量——只有二级位向量位与为1才进行底层位向量对应块的位与运算。还可考虑对DocID进行重排，令位向量更聚集等策略（参见参考文献）。
   2. 多线程：可考虑Query间并行和Query内并行两种方式。前者是指每个Query还是由单线程处理（当然可进行SIMD并行化），不同线程处理不同Query，优点是额外开销少，有利于高吞吐率，但不能优化响应延迟；后者是将一个Query的处理进行任务划分，由多个线程共同完成，优点是能优化大Query的响应延迟，但有额外开销，不利于高吞吐率。当然，仍可考虑位图存储方式的多线程优化。
   3. MPI：思路上类似多线程并行，但由于多节点并行，通信开销远大于多线程并行，这是需要考虑的地方。
   4. GPU：线程块内（特别是warp内）近似SIMD并行，可参考其设计思路。线程块间类似多线程并行，可参考其设计思路。同时考虑GPU特殊的架构特点进行调整。
3. 性能评价指标。对于搜索引擎请求处理问题，主要考虑两个指标，一是响应延迟，即一个Query处理所需时间，另一个是吞吐率，即每秒Query处理量。当然，并行算法与串行算法对比，加速比、效率乃至等效率函数等指标的分析也是很重要的。
4. 在数据集上进行对比实验（对ExpQuery中所有查询请求，进行列表（来自于ExpIndex）求交运算，计算平均响应时间和吞吐率），分析结果，撰写研究报告。

### 4、参考文献

倒排索引、请求处理的基本知识及并行化思路可参考《多核/众核平台下索引压缩及集合求交并行算法研究》（敖耐勇，博士学位论文，2016）。有余力的同学还可研究其中倒排索引压缩及其上的请求处理算法和并行化方法。

## 选题示例四——Gröbner基计算中的高斯消去（算法研究类）

### 1、问题描述及数据集描述

参见《实验教学指导书-3》。

数据集百度盘链接: https://pan.baidu.com/s/1WiFWEplE7KaNaJpZzgOJgA 提取码: xqv1 复制这段内容后打开百度网盘手机App，操作更方便哦

数据集鲲鹏服务器目录：/home/sTest/data/Groebner

### 2、研究思路参考

1. 数据结构。可采用位向量方式存储每个消元子和被消元行，优点是消元操作变为位向量异或操作，算法实现简单，且适合并行化，易达到更高的并行效率，缺点是Gröbner基计算中产生的消元子和被消元行非常稀疏，非零元素（1元素）在5%以下，位向量存储和计算可能并非最优。参考选题示例三，消元子和被消元行可采用类似倒排链表的存储及方式——可认为是稀疏0/1矩阵的紧凑存储方式，每个消元子和被消元行只保存1元素的位置，且按升序排列，从而类似倒排链表数据结构。优点是存储空间占用更少，缺点是算法设计更复杂，并行化难度高。
2. 数据访问顺序。如《实验教学指导书-3》所述，矩阵规模可能非常庞大，达到数百万行/列，难以全部放入内存。此时，需要设计的就不是同学们之前主要接触的内存算法，而是外存算法。需要考虑如何分批次将数据读入内存进行处理，同时又保证正确性。《实验教学指导书-3》中介绍了简单的批次划分方法和数据访问顺序，在期末研究报告中可考虑其他可能的批次划分和访问顺序，以获得更高性能。注意，对外存算法，算法分析和时间测试除了考虑计算之外，还要考虑I/O时间。
3. 消元顺序（数据访问顺序与之相关）。《实验教学指导书-3》描述了一种简单的按行号顺序进行消元的方法，可考虑其他可能的顺序，以获得更好的性能。
4. 并行化方法。
   1. 对位向量存储方式，两行间消元操作的并行化很直接，无论SIMD、多线程还是MPI、GPU，将向量拆分，子向量的异或即自然形成任务，可分配给不同的计算单元。当矩阵规模大到百万级别时，采取这种并行方式就够了。但当矩阵规模没有那么大时，一个消元操作计算量不足以支撑较大规模并行，就需要考虑消元操作间的并行，此时就需要结合3)、2)的消元顺序来设计适合的并行任务划分，在提高并发度的同时保证正确性。
   2. 对类倒排链表存储方式，可参考选题示例三中的讨论。
   3. 当矩阵规模非常庞大，需使用外存算法时，并行算法的设计就要同时考虑计算和访存。例如，多线程并行化时可考虑计算和访存异步模式，在前台线程进行消元计算时、后台线程读取下一步要处理的数据，此时就要仔细设计计算和I/O步骤，降低依赖关系，以便实现异步模式。