《并行计算》
结课报告

姓名：_____

学号：_____

时间：25/5/4

成绩：_____

评分：_____

审核：_____

天津大学 智能与计算学部

2025 年

一、实验内容概述

1.1 实验环境

实验 1，2，3 的环境相同，为天河超算，具体来说，实验环境的一些系统参数为：

(1)操作系统：国家超级计算天津中心定制操作系统(2)CPU：国产飞腾处理器(3)网络：天河自主高速互联网络（400GB/s）(4)处理器性能：单核理论性能（双精度）9.2GFlops，单节点理论性能（双精度）588.8GFlops。

编译环境为：

(1)GCC：版本 9.3.0，包括 gcc，g++，gfort 等(2) OpenMPI：版本 4.1.1，包括 mpicc，mpic++等

对于实验 4，环境是超算互联网提供的异构计算环境，具体来说，系统参数为：

(1)CPU：Hygon C86 7285 32-core Processor (2)内存：128GB DDR4 (3)计算网络：200Gb IB (4)主频：2.0GHz (5)显存：16GB HBM2 (6)单卡性能数据：FP64:10.1Tflops (7)加速卡：4*异构加速卡 2，每张卡 64 个 CU，每个 CU 处理 40 个波前，每个波前最多处理 64 个线程。(8)计算架构：gfx906

编译环境：

(1)GCC：版本 7.3.1，包括 GCC，G++等(2) Clang/LLVM：版本 14.0.0，专为 Hygon DCU 定制的版本，包含 HAS。(3)HIP：版本 5.4.23191。(4) ROCm 组件：版本 1.1.9

1.2 实验一

对于实验 1，只要求熟悉使用最基本的实验环境进行实验，包括如何提交作业，如何查看队列，如何编写脚本文件等，在整个实验的考核中占比不大，并且给出了一个参考实现。

实验 1 的具体任务是多线程计算正弦值，使用 pthread 编程框架，方法是利用泰勒级数展开式来计算。

数据分析方面不做要求。

1.3 实验二

对于实验 2，要求使用 pthread 实现矩阵乘任务，学习 pthread 库的基本使用方法，通过软件角度与算法角度进行优化，例如调整计算顺序、循环重排，进行矩阵分块等。数据分析时要计算加速比，效率的可视化。

1.3 实验三

对于实验 3，要求使用 MPI 编程进行实验 2 的任务，即通用矩阵乘，优化性能时可以根据实验 2 用到的方法参考。

1.4 实验四

对于实验 4，要求使用 DCU 编程实现通用矩阵乘，此时任务的拆分可以更加密集，使用到的线程数可以更多。数据分析同样要计算加速比，效率的可视化。

每个实验要求给出程序的流程图，具体算法的伪代码，以及具体的编程实现等。

二、并行算法分析设计

2.1 PThread 多线程算法设计

使用 pthread 编程实现矩阵乘的主要任务在于划分方式，基本的划分方式包括行划分，块划分。

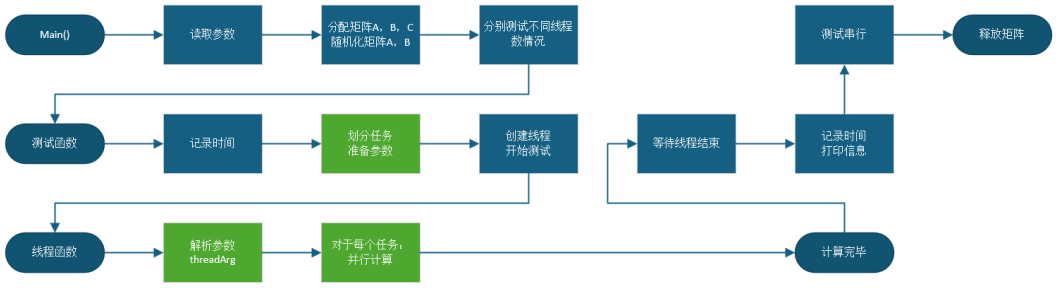

图 2-1　pthread 编程的流程图

由图 2-1 所示，使用 pthread 编程实现矩阵乘的基本流程图对于行划分，块划分只有标注绿色的步骤有差别，其他步骤大致相同。

2.1.1　行划分设计

将输出矩阵 C 的行分配给不同线程，每个线程计算 A 的一部分行和 B 的所有列的乘积，如图 2-2 所示：

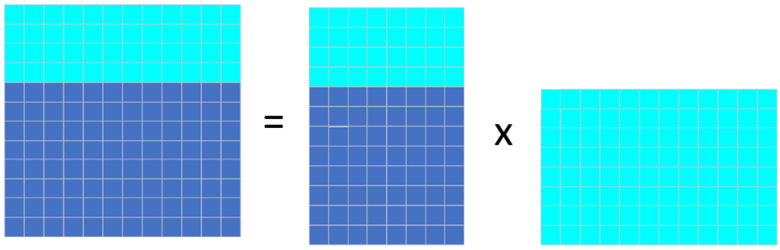

图 2-2　行划分设计示意图

行划分的伪代码如下，主要的就是计算行区间：

```
# 计算初始行区间
args.start_row ← base_length * j
args.end_row ← args.start_row + base_length - 1
# 边界条件处理：确保最后线程不越界
if j == num_threads - 1 then:
    args.end_row ← matrix_rows - 1
```

2.1.2　块划分设计

固定块的大小，将矩阵划分为子块，线程以块为单位计算局部结果，如图 2-3 所示：

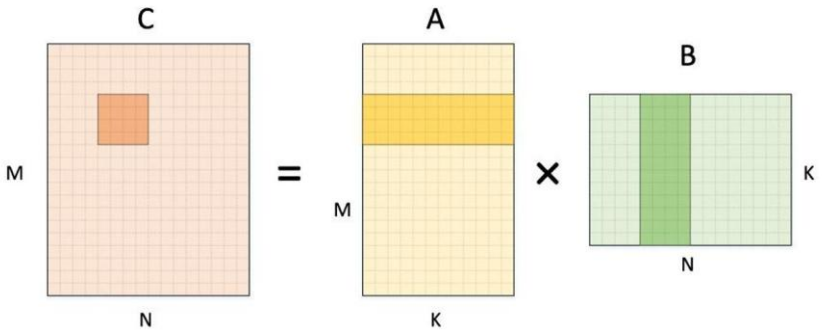

图 2-3　块划分设计示意图

块划分的伪代码如下：

(1)主线程任务划分：

```
num_blocks_m = (矩阵行数M + BLOCK_SIZE - 1) // BLOCK_SIZE  # 行方向分块数
num_blocks_n = (矩阵列数N + BLOCK_SIZE - 1) // BLOCK_SIZE  # 列方向分块数
total_tasks = num_blocks_m * num_blocks_n                  # 总任务数=块矩阵元素总数

tasks_per_thread = total_tasks // n_threads                # 基础任务数
remainder = total_tasks % n_threads                        # 剩余任务分配

current_task = 0
for q in 0..n_threads-1:
    # 计算当前线程的任务区间
    end_task = current_task + tasks_per_thread + (q < remainder ? 1 : 0)

    # 设置线程参数
    thread_args[q] = {
        start_task: current_task,       # 起始任务编号
        end_task: end_task,             # 结束任务编号（不包含）
        num_blocks_n: num_blocks_n,     # 列方向块数（用于坐标转换）
        ...其他矩阵参数...
    }
```

(2)对等线程将一维的任务号转化为二维的区间：

```
# 一维转二维块坐标（行优先）
bi = task_id // args.num_blocks_n  # 块行坐标
bj = task_id % args.num_blocks_n   # 块列坐标

# 计算实际行列范围（处理边界）
i_start = bi * BLOCK_SIZE
i_end = min(i_start + BLOCK_SIZE, M)

j_start = bj * BLOCK_SIZE
j_end = min(j_start + BLOCK_SIZE, N)
```

2.2 MPI 并行算法设计

也有两种实现方法，即行划分、块划分。不同的是，对于多线程，每个线程共享相同的内存空间，多进程的每个进程并不是共享内存的，要通过消息传递来发送数据。

2.2.1 行划分

流程图如图 2-4 所示：

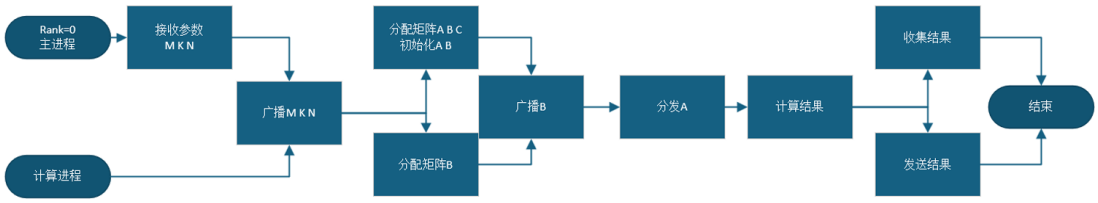

图 2-4　行划分流程图

2.2.2 块划分主从模式

将可用于计算的进程数 comm_sz 分解为 a*b，然后将矩阵 A 全体行划分为 a 个部分，将矩阵 B 全体列划分为 b 个部分，从而将整个结果矩阵划分为 size 相同的 comm_sz 个块。每个子进程负责计算最终结果的一块，如图 2-5 所示：

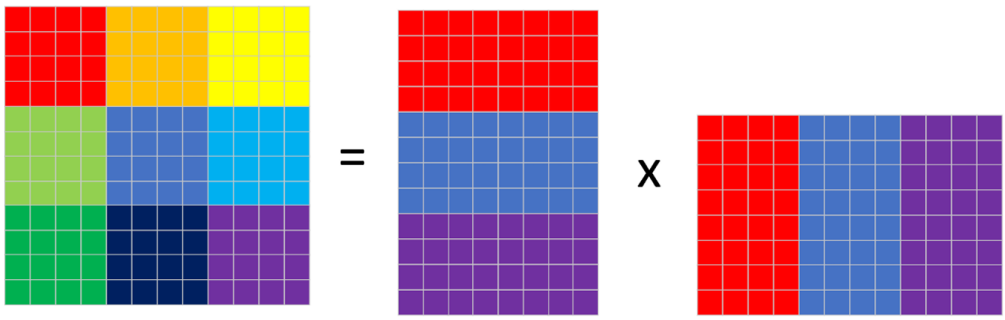

图 2-5　主从模式块划分

在这种任务分配模式下，所有的进程分为两个角色，主进程进行分发任务和数据，收集与整合结果，从进程负责接收分配的任务，局部计算，将结果发送到主进程，因此，叫做主从模式。

流程图如图 2-6 所示：

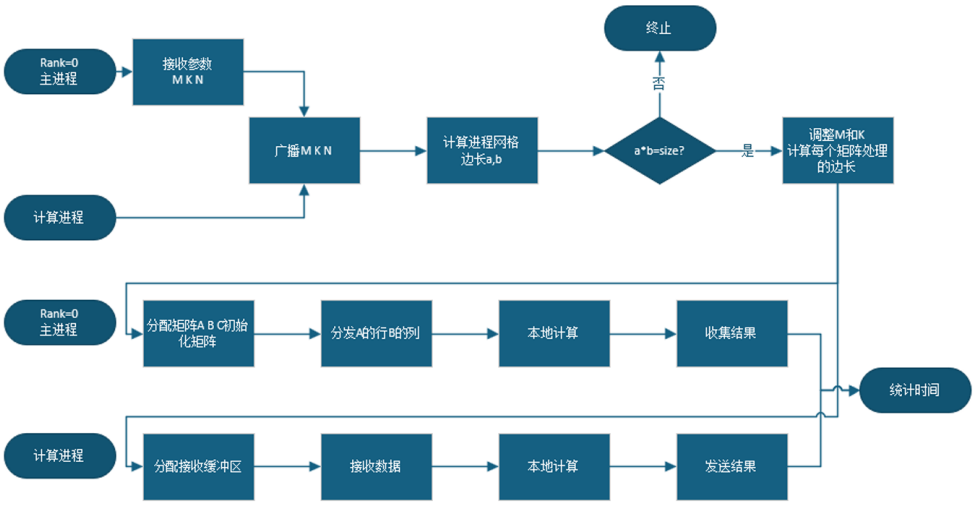

图 2-6　主从模式流程图

2.2.3　块划分对等模式

将可用于计算的进程数为 comm_sz，然后将矩阵 A 全体行划分为 comm_sz 个部分，将矩阵 B 全体列划分为 comm_sz 个部分，从而将整个结果矩阵 C 划分为 size 相同的 comm_sz*comm_sz 个块。每个子进程负责计算最终结果的一块，然后循环交换 B 的列块，进行下一次计算，直到完成，每个进程访问完毕 B 的所有列块，即块轮转策略，示意图如图 2-7 所示：

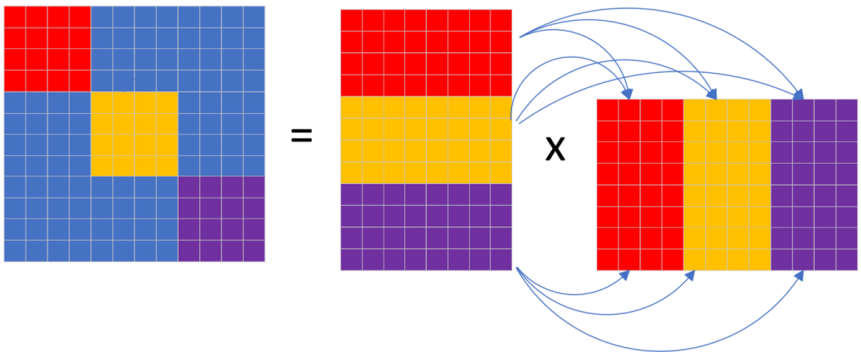

图 2-7　块轮转计算矩阵乘

流程图如图 2-8 所示：

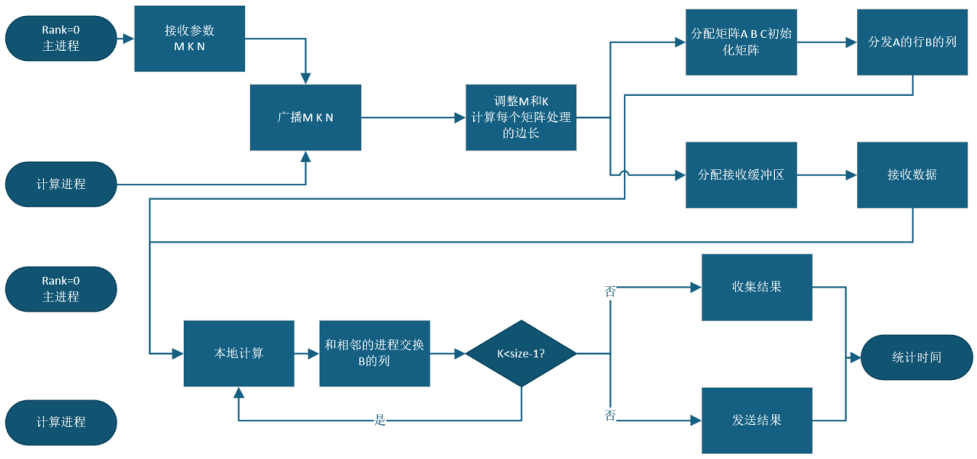

图 2-8　块轮转流程图

2.2.4　非阻塞通信

实现上述方法的非阻塞通信的版本，最大程度上的并行化消息传递，由于矩阵乘任务需要将数据就位后才能开始计算，因此，使用非阻塞的版本只能将多个消息传递并行化，而不能发挥一边通信，一边计算的优势。

2.3 异构并行算法设计

如果不使用共享内存进行优化，那么使用 GPU 异构编程时，将矩阵拷贝到 GPU 的显存中之后，就可以像 CPU 多线程那样，直接访问对应的元素，不需要进行通信。

流程图如图 2-9 所示：

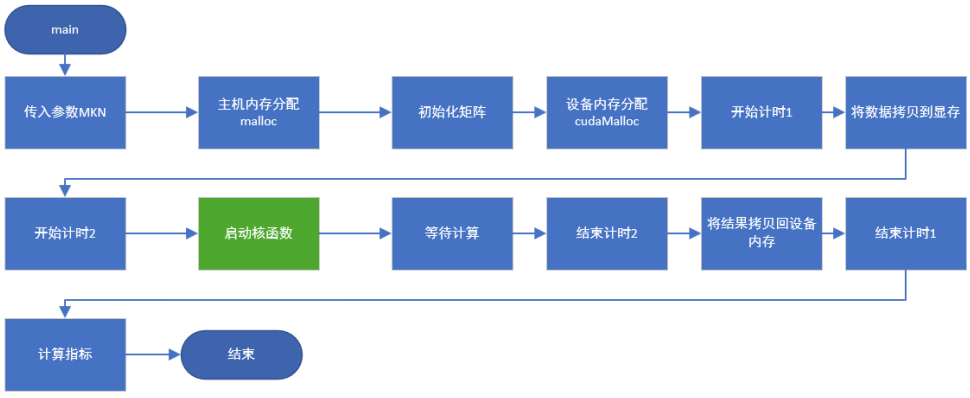

图 2-9　DCU 编程实现矩阵乘的流程图

不同实现方法的核函数不同，其他的内容，包括分配内存，计算指标这些内容都相同。

2.3.1　1 维块构建

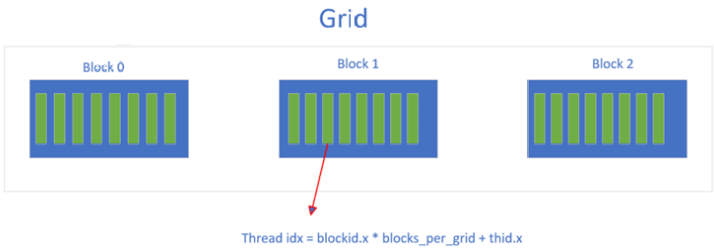

图 2-10　一维线程索引计算

由图 2-10 所示，采用 1 维块进行矩阵乘的计算时，根据每个线程的索引计算出对应的结果矩阵的行列，然后局部计算当前值。

$$Cx = thID / wC$$
$$Cy = thID \% wC$$

2.3.2 2 维块构建：

使用 2 维块的索引计算方式与 1 维块不同，如图 2-11 所示：

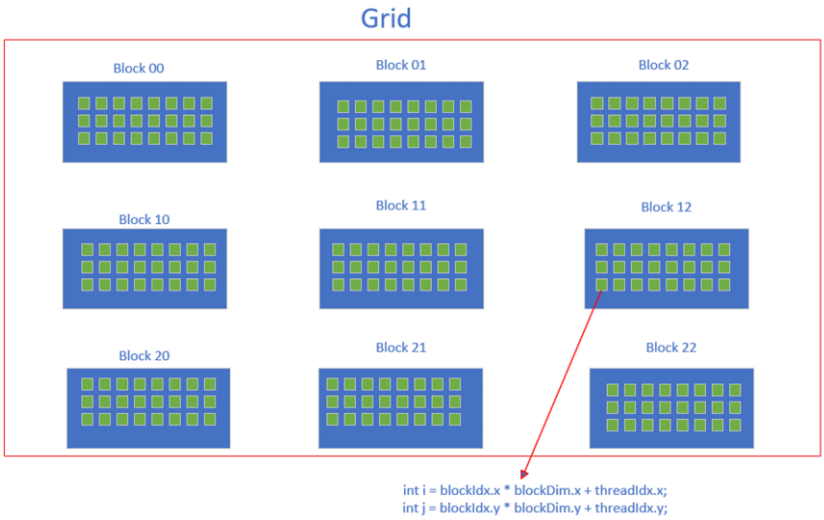

图 2-11 2 维块索引计算

然后每个线程计算 1 个结果矩阵的元素：

```
if m < M 且 n < N:
    sum = 0.0
    for k in 0 到 K-1: # 遍历 K 维做点积
        sum += A[m][k] * B[k][n]
    C[m][n] = sum    # 写入结果
```

2.3.3 共享内存优化

使用上面的方式计算并没有有效的数据复用，在 GPU 中，共享内存是一个 block 内的所有线程共享的，此时，只需要访问一次全局内存，将数据加载到共享内存，就可以降低时间，优化性能。假设共享内存 16KB，正好存下两个 32*32 的 double 类型的数据块，对应 C 的子块。 在 A 中就是 32 行，在 B 中就是 32 列，每次加载 32 个，一次一次，将整行/整列计算完毕。每次计算时，首先将子块加载到共享内存，然后每个线程使用共享内存的数据进行计算，最后将结果写回全局内存。

```
for k_base in 0 到 K 步长 BLOCK_SIZE:
        # 阶段 1: 协作加载 A 的分块到 s_a ----------------------------
        a_row = by + ty          # 全局行坐标
        a_col = k_base + tx      # 全局列坐标
        if a_row < M 且 a_col < K:
            s_a[ty][tx] = A[a_row][a_col]  # 行优先加载
        else:
            s_a[ty][tx] = 0.0  # 边界填充
        # 阶段 2: 协作加载 B 的分块到 s_b ----------------------------
        b_row = k_base + ty      # 全局行坐标
        b_col = bx + tx          # 全局列坐标
        if b_row < K 且 b_col < N:
```

```
            s_b[ty][tx] = B[b_row][b_col]  # 行优先加载
        else:
            s_b[ty][tx] = 0.0  # 边界填充
        同步块内所有线程  # 等待数据加载完成
        # 阶段 3: 共享内存矩阵乘累加 ----------------------------
        for kk in 0 到 BLOCK_SIZE-1:
            # s_a 按行访问，s_b 按列访问（矩阵乘法则）
            psum += s_a[ty][kk] * s_b[kk][tx]
```

三、实验数据分析

3.1 实验环境

见 1.1 节

3.2 实验数据综合分析

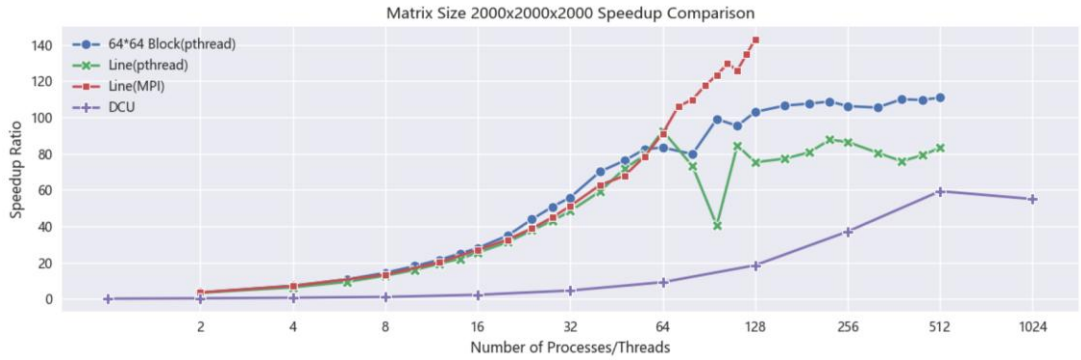

图 3-1　2000 规模下不同编程实现加速比对比

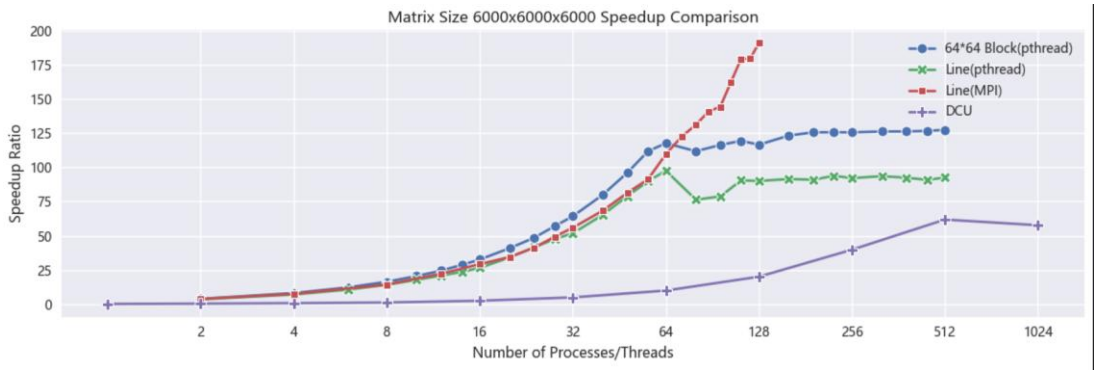

图 3-2　6000 规模下不同编程实现加速比对比

由图 3-1，3-2 可知，使用块划分的多线程实现在核数少于 64 时要好于 MPI 多进程，这是因为线程的创建非常快捷，开销很小，因为内核无需单独复制进程的内存空间或文件描述符等等，从而节省了时间，而使用行划分的多进程，多线程开销基本一致，说明在单节点中，数据的传输速度很快。另一方面，使用多线程的性能上限较低，这是因为它受限于单个物理节点，在实验环境下，一个物理节点含有 64 个 CPU，因此，在线程数继续上升时，加速比保持平稳。对于 MPI 多线程，可以实现跨节点通信，可扩展性强，使用 2 个节点的情况下，创建多于 64 个进程，加速比可以继续提升。

而对于 DCU，由于显卡内的单个线程的开销极小，远小于 CPU 的线程，因此，当线程数相同时，得到的加速比很低，而 DCU 只有在更多的线程下，以合理的方式组织 block，性能才能有较好提升。

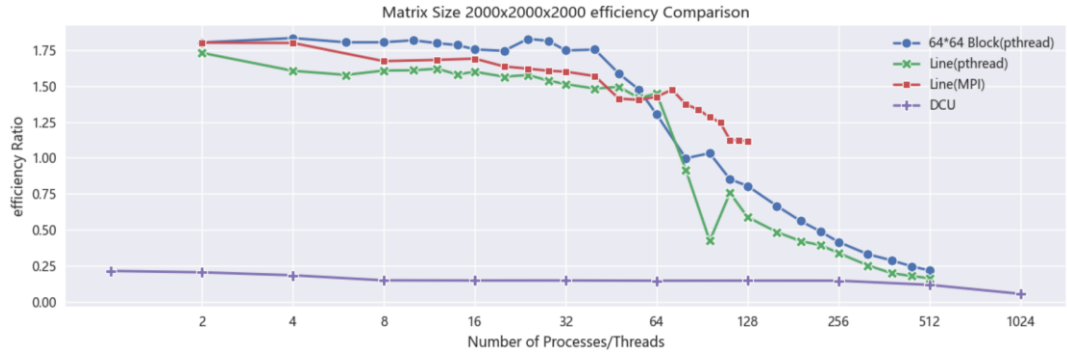

图 3-3　2000 规模下不同编程实现的效率对比

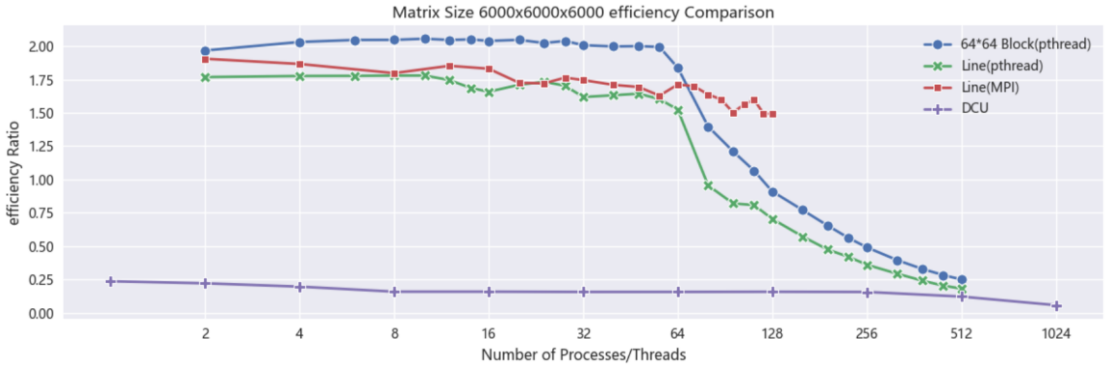

图 3-4　6000 规模下不同编程实现的效率对比

由图 3-4 所示，使用的核数多于 64 时，pthread 编程实现的效率急剧下降，因为每个节点只有 64 个核，继续增加线程数，不会提高任何性能，而对于 MPI 多进程，由于较好的可扩展性，因此效率虽然受到了跨节点通信延迟增加而下降，但是幅度小于 pthread 实现。对于 DCU 实现，由于使用的线程数较小，每个 block 内只有一个线程，无法充分的利用 DCU 的计算资源，因此，效率较低。图中数据点的效率大多数都大于 1，这是因为使用的 CPU 基准串行时间是没有访存优化的，而在进行 pthread 和 MPI 实现时，具有循环重排以提高局部性，因此，效率大于 1。

为了更好的体现出 DCU 的性能，下面画出不同规模下，使用不同编程实现，效率/加速比最大的数据点，而不再考虑进程/线程数。

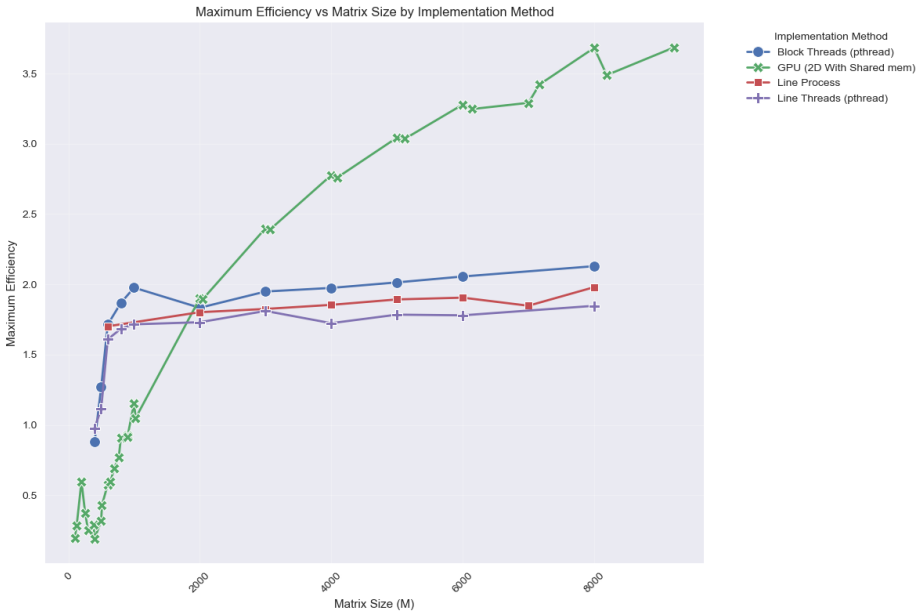

图 3-5　不同规模下最大效率

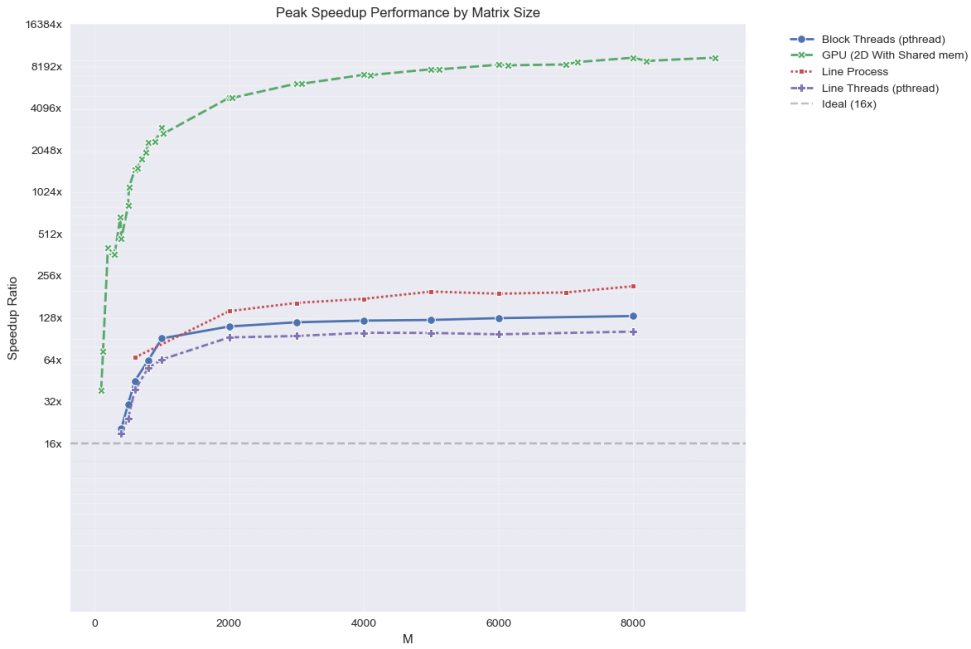

图 3-6　不同规模下最大加速比

由图 3-5，3-6 所示，如果不限制使用的进程/线程数，在矩阵规模较小时，加速比和效率较低，这是因为此时在整个计算任务中，矩阵运算占比低，线程/进程总开销占比大；随着矩阵规模增大，线程/进程开销比例降低，加速比和效率提高，并在规模大于一定数值时，保持相对稳定，可扩展性在一定程度上受到限制。对于 DCU，由于其特殊架构，专门的非常适合计算矩阵乘，因此，会达到极高的效率和加速比，还有一方面原因可能是因为效率计算的方法，除以的波前数，可能与实际有一些偏差。

四、实验总结

4.1　问题总结

(1)由于进行实验的环境都为类似 linux 的操作系统，大部分地依赖命令行，每次在自己的 windows 电脑上改一行代码就要很麻烦的重新传输一次文件，因此不得已之下学习了 vim 的一些基本操作，并且已经

能够熟练运用，通过本课程的实验，也让我更加熟练的使用 linux 系统。

在实验 4 的平台上由于和之前的超算作业提交的方式有区别，因此摸索了很久，以及通过 ai 才了解到要使用 DCU 编译要先加载环境，以及使用特定的编译选项，才可以正常提交 DCU 的任务，否则只要执行到 hip 代码，就会段错误。通过这一困难，我学习到了更多的关于 DCU 编程，以及编译的知识，也了解到如何编写并行计算任务的脚本，包括申请资源等等。

(2)编写代码的时候由于经验不足，有时会出现难以调试的 bug，这时可以借助于 ai 工具，由于最近 ai 的不断发展，能力也有了很强的提升，除了可以修改 bug，还可以帮助编写代码。比如，编写行划分是比较容易的，但是此时，如果还想要进一步的探究，编写一个块划分的版本，如果直接写，对于编程的要求就很高了，使用 ai 就可以完成很多工作，因此实验探究的内容就可以有很多，可以加深自己的理解。例如，使用 pthread 实现矩阵乘的实验中，如果要使用块划分，具体的设计方法是根据 ai 学来的，也就是固定一个分块大小，任务划分阶段将一维的任务号传入线程函数，然后在线程中将一维任务号转化为二维索引范围，进行局部计算。之前的想法一直是要把线程数开根，然后分块计算，但是并不好下手。

在进行实验 3，MPI 编程实现矩阵乘的时候，初始想法是和实验 2 的块划分方法相同，也是固定块大小，但是在实际编程之后的测试中，发现死锁，问题也没有排查成功，因此，采用了线程数开根，分块计算的方法，虽然这样可以运行，但是只能限定特定的进程数。

(3)在进行实验 4 的时候，为了契合编写的矩阵乘代码，想要测试一下 2 的幂次情况的矩阵规模情况，但是前面的实验 2 仅测试了整百情况的串行基准，此时要重新计算串行时间可能需要的时间较多，如何不进行测试，得到其他规模的串行时间？既然是串行，那么时间复杂度就是 $O(n^3)$，可以考虑使用线性回归的机器学习算法，根据已有的数据，进行预测。首先将数据转换为对数，这样就可以使用线性回归，发现性能很好，已有的数据完全符合线性分布，因此这种方法可行。

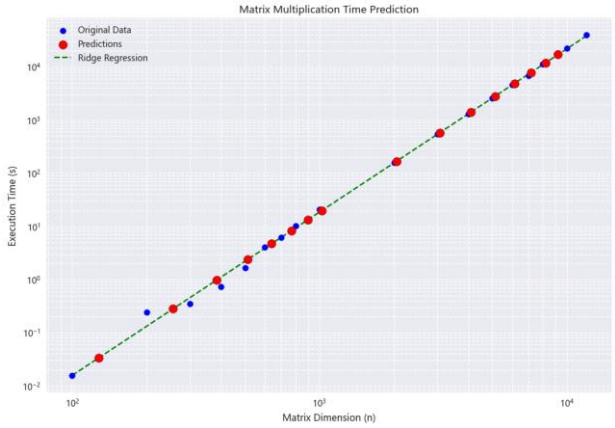

图 4-1　预测串行时间

4.2　不同并行计算的分析

(1)多线程：

同一进程内的多个线程共享内存空间和文件描述符，通信高效，线程创建和调度的开销较低，适合频繁的任务调度。因此具有容易部署，编程相对简单，通信延迟低，不需要显式实现，任务划分粒度可以自由调节，负载均衡实现起来容易。

但是多线程之间的共享内存仅限于单个进程内部，受限于物理机，可扩展性局限于单个节点内部增加 cpu 核数等，并且线程之间的同步较麻烦，一旦某个线程崩溃，会影响到整个的进程，在实验 2 中，使用 pthread 实现的矩阵乘在线程数小于 cpu 数时，性能较好，但是线程数一旦超过 cpu 数，那么加速比就保持稳定，不再上升，因此受限于单节点，由图 3-2 所示。

(2)多进程

每个进程拥有独立内存空间，通过进程间通信交换数据。由于进程之间可以相互通信，就大大增加了

可扩展性，运行在不同物理机上的进程也可以通过网络通信，突破了线程受限于单个物理节点，在实验 3 中，使用 MPI 实现的矩阵乘，在进程数大于单个节点的 64cpu 时，加速比依然有提升，此时该任务用到了 2 个节点，由此可以证明多进程的可扩展性，由图 3-2 所示。

但是使用 MPI 多进程编程，通信一般会跨物理节点，难以全面的了解系统状态，因此调试更加困难，节点之间的通信也有可能发生超时，这些不确定的情况。另一方面，由于使用 MPI 多进程编程需要自己实现进程间的消息发送，接收，编程的难度也更大，主进程由于需要不断的发送任务，以及接收数据，相较于从进程，负载更多，进程之间的负载均衡实现也较为困难。

(3)异构计算

异构计算指计算系统中同时包括多种不同架构的处理器，既有 CPU，也有 DCU，GPU 这些加速部件，相较于 CPU，DCU 这些部件计算环境本身都是专为并行计算设计，计算芯片占比极大，控制芯片占比小，因此非常适合进行矩阵相乘这样的任务。使用异构计算可以将逻辑判别计算调度到 X86 架构的 CPU，而将需要能够大规模向量并行的计算部分调度到 GPU，最大的发挥不同计算平台的优势。以 GPU 为例，线程调度分为 3 个层次：grid，block，thread，其中，可以派生的线程数极多，以实验 4 的共享内存实现为例，如果共享内存的大小是 32*32，那么每个线程只需要计算一个元素，每个 block 计算最终结果矩阵 C 的一个 32*32 的块，因此，一个 block 中含有 32*32=1024 个线程，对于 8000 规模的矩阵，就要创建 250*250=62500 个 block，总共的线程数就是 1024*62500=64000000 个线程，规模巨大，远超 pthread 与 MPI 的编程，如图 3-6，如果不限制最终派生的线程个数，那么使用 GPU(DCU)编程实现的加速比是 pthread 与 MPI 的几十上百倍。由于可以派生的线程数量极大，因此每个线程的开销就极低，如果限制派生的线程数，由图 3-2 可知，DCU 的线程与其他两种编程实现相比，在数量相同下，性能很低。

DCU 编程由于是跨平台的，因此需要学习的架构就有很多，并且编写的代码也不能有过多的逻辑判断的内容，编程的难度较大。

五、课程总结

5.1 本学期授课的优点

(1)老师上课的态度非常认真负责，对于课件的讲解也很生动形象，总的来说，老师上课时的讲课方式很好，我认为足够吸引我认真听课，让我学到知识。

(2)理论教学方面，ppt 上的知识点展现形式比较直观，结合老师上课的讲解，可以很清晰的看懂，比如在并行算法设计这个 ppt 中，并行求前缀和部分的配图很清晰，并且每个步骤一页一页的展现，而不是一页直接打出所有的结果，在进行回顾的时候很有帮助。

课程的整体逻辑也比较清晰，层次也很分明，整个课程除去概述方面，先是介绍了并行计算应的必要性，在什么硬件上进行，然后介绍了 pthread 编程，以及评判并行程序性能的一些指标，包括加速比，效率。接着介绍了 openMP，即使实验没有用到，但是这个技术在这门课之外也是很重要的，我觉得这很好，虽然没有进行实验，但是很好的扩充了我的知识体系。然后介绍了 MPI 编程，包括阻塞，非阻塞的通信方式，以及异构计算。之后进入到课程的后半阶段，介绍了多线程与多进程这两种编程的对比，以及如何将这两者结合，然后是对于数据密集型任务如何进行并行化，最后对课程总结，包括如何设计并行计算的算法等。整体学习下来，由于大部分是进行介绍，因此没有感觉到很困难的部分，授课节奏把握也很好。

(3)实验方面，我认为，单看实现矩阵乘这个任务比较简单，因此也有更多的精力探究不同的方式实现的性能对比是怎样的，比如，对于实验 2，我用 pthread 实现了行划分，块划分两种，在实验 3，我实现了行划分，主从模式块划分，对等模式(块轮转)块划分，以及这三种的阻塞，非阻塞版本，以深入探究他们的性能对比，在实验 4，为了可以和实验 2，3 进行横向对比，先是实现了一个使用特定线程数实现的版本，但是根据 DCU/GPU 的特点，很适合派生极大数量的线程，因此为了进一步探究 DCU 的性能，我又不限制线程数，实现了使用 2 维块的版本，以及使用共享内存进一步优化访存的版本。总的来说，就是因为矩阵乘这个任务比较简单，所以我有精力以较多的方式来实现，因此，我觉得，使用矩阵乘这个任务作为上机实验也是很好的。

5.2 课程的建议

(1)授课时的代码讲解：

老师上课进行讲解时，对于具体代码的讲解较少，例如，在多线程编程部分，对于 pthread 库的讲解就要求自学，这里要求自学的难度其实不大，并且在《计算机系统基础 2》课程中也已经学习了 pthread 的一部分。但是在后面异构计算的部分中，具体如何编写核函数，这一部分也要求了自学，对于这一部分，先前并没有了解，因此，自学的难度有一定难度，希望老师可以先在课上进行一些基本的讲解，这样即使后续需要自学，学起来难度也会降低。

(2)每次授课的作业：

本课程除了上机实验。还有一个要完成的部分就是每次教室上课之后留的作业，这一部分大多是一些上课知识点的整理与分析。这部分作业在之前 ai 没有发展起来确实可以增加同学的理解，但是在现在的这个阶段，使用 ai 大模型就可以很好的给出答案，因此可以调整一下，换一些具体的作业题，而不是问一些例如 mapreduce 编程模型的优点这样的很适合 ai 回答的问题，因为这些问题使用 ai 都可以得到很好的答案，所以老师可以默认同学们对于这样的问题都会使用 ai 来了解，因此可以不再问这些问题。转而去问一些 ai 不容易回答的问题，例如第三次作业的问题四：“4、根据第 62/63 页 PPT 的代码，用文字描述当 i 从 1 到 4 时两个线程加锁解锁的动作时间序列。”这样就可以让同学们自己探究，并给出回答。

另一方面，也可以进行一些小的实验，比如，上机的实验要求 pthread，MPI，DCU，那么在平时作业就可以安排一些其他的编程模型的小任务，例如 OpenMP，CUDA，或者 pthread，MPI，DCU 这些。就像 MIT 的操作系统课程一样，包含两个部分，一个是 homework，每个 1-2h，是一些简单的编程任务，而 lab 的任务相对较大，需要 3-4h，这样即使同学们使用 ai，但是最关键的是自己动手，可以学到很多内容。

每个任务可以给出一个大框架，然后具体的一些任务加上 TODO，表示在这里填写代码：

```
124             if self.L1:
125                 dist_correct = # 【TODO】使用L1范数计算正例距离
126                 dist_corrupt = # 【TODO】使用L1范数计算负例距离
127             else:
128                 dist_correct = # 【TODO】使用L2范数计算正例距离
129                 dist_corrupt = # 【TODO】使用L2范数计算正例距离
130
```

(3)异构编程的性能指标计算：

一开始的授课讲解了对于并行计算性能指标的计算方式，包括，加速比，效率。但是在进行实验 4，DCU 编程的时，由于 DCU 的特性，需要派生极大数量的线程才能得到很好的性能，然而，对于效率应该如何计算？直接使用加速比除以创建的总线程数吗，这能表示出 DCU 的线程与 CPU 的进程，线程的比较吗？既然效率是加速比除以 CPU 数，那么，我在实验 4 中使用到的就是 DCU 一次 SIMD 指令涉及到的线程数，即波前为单位，通过查询，发现也有使用流多处理器(SM)的，所以应该怎么办？希望后面再开本课程可以在上课的时候给出方法，否则，按照前面学习的内容，好像只能不断调整线程数，和 pthread/MPI 对比，但是问题是 DCU 适合极大量的线程派生，不是很适合这种办法。

所以具体应该怎样对比？我也不太清楚，我想的方法，也就是画出不同规模下最大的加速比，而对于效率并没有讨论较多，我并不清楚这个方法是否合理。

(4)上机实验：

对于上机实验方面，同样由于最近 ai 大模型的急速发展，我认为，光写一个很简单的行划分的实现，有点不够，通过 ai 的加持，可以实现一些更加复杂的，比如块划分等等，所以我觉得，可以更加进一步具体一下实验的要求，比如说，要实现哪些划分方式，以及采用什么方法等等。

另一方面，平时授课的时候，也可以讲一下实验的一些可行的方案，比如行划分，块划分的这些。

附：上机实验与课程知识点分析

序号	上机实验内容	理论知识点	分析总结
1	实验 2 pthread 实现矩阵乘	多线程并行程序设计	实验 2 需要用到 pthread 实现矩阵乘，此次授课讲述了进行多线程设计的一些注意点，包括 pthread 库，对于共享内存如何访问，以避免假共享
2	实验 2，3，4 中数据分析部分	并行计算的性能	实验 2，3，4 进行数据分析过程中，需要计算加速比，效率，此课件描述了对于加速比，效率的计算方法，比如，对于加速比就是串行执行时间除以并行时间。以及，对于可扩展性的评测，即并行计算性能指标随处理器数增加而按比例提高的能力
3	实验 2，3，4 作业提交部分	MPI 基础 ppt 中的并行计算作业管理部分	在该课件中，介绍了并行计算的不同作业管理系统，包括 slurm，和 PBS。其中，无论是实验 2，3 的天河超算，还是实验 4 的异构计算平台，使用的作业管理系统都是 slurm，对应了介绍到的 slurm 的架构，部署，资源分区，以及资源申请，作业提交的命令
4	实验 3MPI 实现矩阵乘，使用阻塞通信	MPI 基础 ppt 中的 MPI 介绍部分	在课件中，该部分介绍了使用 MPI 编程的 6 个基本接口，以及基本的使用方法，点对点通信的 MPI_Send，MPI_Receive，组通信中，一对多，多对一等等，而在实验中，我实现了行划分，这用到了 MPI_Bcast 等，实现了块划分，用到了 MPI_Send 等，为了同步计时用到了 MPI_Barrier，MPI_Reduce 等，这对我的实验有很大帮助
5	实验 3MPI 实现矩阵乘，使用阻塞通信	MPI 基础 ppt 中的阻塞通信模式	该部分介绍了一些基本的阻塞通信模式，包括标准通信模式，缓存通信模式等，虽然在实验中使用的标准通信模式，但是其他的通信模式也有各自的特点与适用情况
6	实验 3 中使用非阻塞通信实现矩阵乘	MPI 进阶 ppt 中非阻塞通信介绍	该部分介绍了一些基本的非阻塞通信方法，以及一些专门的语句用于完成或者等待非阻塞通信，在实验 3 中，用到了一些非阻塞的通信函数，例如 MPI_Iscatterv，MPI_Ibcast，MPI_Isend 等，这对我编写代码很有帮助
7	实验 3 中块划分对等模式使用 MPI_Sendrecv_replace	MPI 进阶 ppt 中 MPI_Sendrecv 介绍	在使用对等模式实现的矩阵乘中，每个进程都要向相邻的进程发送矩阵 B 的子列，在这种情况下，使用 MPI_Sendrecv 可以捆绑发送和接收，避免了潜在的死锁，在实验 4 中，我用到的是 MPI_Sendrecv_replace，在一个原子操作里实现数据的发送和接收，并且用接收到的数据替换发送缓冲区中的数据
8	实验 3 中块划分对等模式的非阻塞实现中使用到了 MPI 自定义数据类型	MPI 进阶 ppt 中自定义数据类型介绍	在该部分，介绍了除去 MPI 中的基本数据类型，还可以自定义数据类型，包括自定义结构体，自定义向量，自定义连续数据，在实验 3 中，使用到了自定义向量，进行 B 矩阵子列的传输
9	实验 4DCU 实现矩阵乘基本思想	异构计算 ppt 中关于 GPU 计算过程介绍	在课件的该部分，介绍了使用 GPU 进行异构计算的基本流程，包括将数据从 CPU 内存拷贝到 GPU 显存，启动核函数计算，将结果拷贝回 CPU 内存。在实验 4 中的具体编程中使用的基本思路都是这样的

10	实验 4 使用 DCU 编程实现的编程语言	异构计算 ppt 中关于 cuda 的介绍	在实验 4 的 DCU 编程中，实际应该用到的是 HIP 编程，但是使用指令 hipify-perl 可以直接将.cu 的 cuda 代码转换为 HIP 代码，因此，课件中介绍的 cuda 代码对于我实验 4 的编程也很有帮助
11	实验 4 编程实现矩阵乘的核函数矩阵访问方式	异构计算 ppt 中关于 Blocks + Threads 的索引数组介绍	在课件的该部分，介绍了如何使用 Blocks + Threads 索引来访问数组，在实验 4 的编程实现中，使用的就是 Blocks + Threads 联合的索引来访问数组，包括 1 维，2 维两种
12	实验 4 编程实现的线程组织方式	异构计算 ppt 中关于线程层次介绍	课件中的该部分介绍了 GPU 中线程的 3 个层次，包括 grid，block，thread，其中在实验 4 的编程实现中，用到了 block 和 thread 两个层次，因此也使用了维度变量
13	实验 4 使用共享内存优化 DCU 矩阵乘	异构计算 ppt 中共享与同步介绍	在课件的该部分描述了共享内存的基本使用方式，以及同步函数__syncthreads()，在实验 4 中用共享内存优化 DCU 矩阵乘用到了这些内容
14	实验 4 的 1，2 维，共享内存实现	异构计算 ppt 中关于性能优化的介绍	该部分介绍了优化 DCU 并行计算性能的基本方法，不同于 CPU，DCU 的线程切换效率很高，因此，程序应该尽可能派生多的线程，以提高并行度。实验 4 除了使用小线程数，还实现了用更多线程的矩阵乘，结果表明使用更多线程确实可以提高性能
15	实验 2，3，4 对于矩阵乘任务的划分	并行程序设计方法学 ppt，并行计算概述 ppt 域分解介绍	课件的该部分介绍了域分解的基本类型与方法，包括规则区域，不规则区域的域分解。其中对于矩阵乘任务，结果矩阵的每个元素没有依赖，因此可以直接划分，实验中，使用到了行划分，块划分这两种方法