并行程序设计与算法 第一次作业

刘森元, 21307289

中山大学计算机学院

1 简答题

1.1 习题 1

为求全局总和例子中的 `my_first_i` 和 `my_last_i` 推导一个公式. 需要注意的是: 在循环中, 应该给各个核分配数目大致相同的计算元素. (提示: 先考虑 n 能被 p 整除的情况).

```
int numPerCore = (n + p - 1) / p;
for (int rank = 0; rank < n; ++rank) {
    my_first_i[rank] =      rank * numPerCore;
    my_last_i[rank] =       min(n, (rank + 1) * numPerCore);
    // [my_first_i, my_last_i) 左闭右开区间
}
```

1.2 习题 2

(1) 解释局部性原理

局部性原理主要分为两个方面:

1. **时间局部性 (Temporal Locality):**

   若一个数据正在被访问, 则近期很有可能会被再次访问. 通常由程序中的循环操作引起.

2. **空间局部性 (Spatial Locality):**

   若一个数据正在被访问, 则近期其邻近的数据很有可能被访问. 通常因为数据是顺序存放引起.

(2) 在以下的代码中, 存在何种局部性

```
float z[10000];
float sum = 0.0;
for (int i = 0; i < 2000; i++)
        sum += z[i];
```

上面的代码主要体现了空间局部性.

1.3 习题 3

(1) 当 CPU 将数据写入缓存时, 缓存中的值可能与主存中的值不一致, 有哪两种解决策略? 请阐述.

该问题即缓存不一致性问题. 主要解决策略有:

1. 写直达法 (Write-Through):

   CPU 每次写入缓存, 同时也会将数据写入主存

2. **写回法 (Write-Back):**

   CPU 只将数据写入缓存, 只有当缓存块需要被替换或者发生失效等情况时, 才将修改后的数据写回主存.

(2) cache 映射的方式有哪三种? 请阐述.

1. **直接映射 (Direct mapping):**

   主存块直接映射到 Cache 的一个固定位置..

2. **全相联映射 (Fully associative mapping):**

   主存块映射到 Cache 的任意一行.

3. **组相联映射 (Group associated mapping):**

   Cache 被分为若干组, 每组包含若干块, 对应主存中数量相同的块, 主存块可任意映射到对应块.

1.4 习题 4

在冯 · 诺伊曼系统中加入缓存和虚拟内存改变了他作为 SISD 系统的类型吗? 如果加入流水线呢? 多发射或硬件多线程呢?

1. **缓存和虚拟内存:**

   没有改变其作为单指令单数据流 (SISD)系统的基本类型. SISD系统一次只处理一条指令和一个数据流, 而缓存和虚拟内存主要影响的是数据的访问和存储方式, 不改变指令的执行方式.

2. **流水线:**

   没有改变冯 · 诺伊曼系统作为SISD系统的基本属性. 流水线只是改变了指令的执行方式, 使得多个指令可以同时执行, 但仍然是一次只处理一条指令和一个数据流.

3. **多发射或硬件多线程:**

   改变了其作为SISD系统的基本类型. 多发射可能使系统变为单指令多数据流 (SIMD)系统, 而硬件多线程则可能使系统变为多指令多数据流 (MIMD)系统.

2 计算题

2.1 习题 5

在下列情况中, 推倒求出 0 号核执行接收与加法操作的次数 (假设一共有 p 个核).

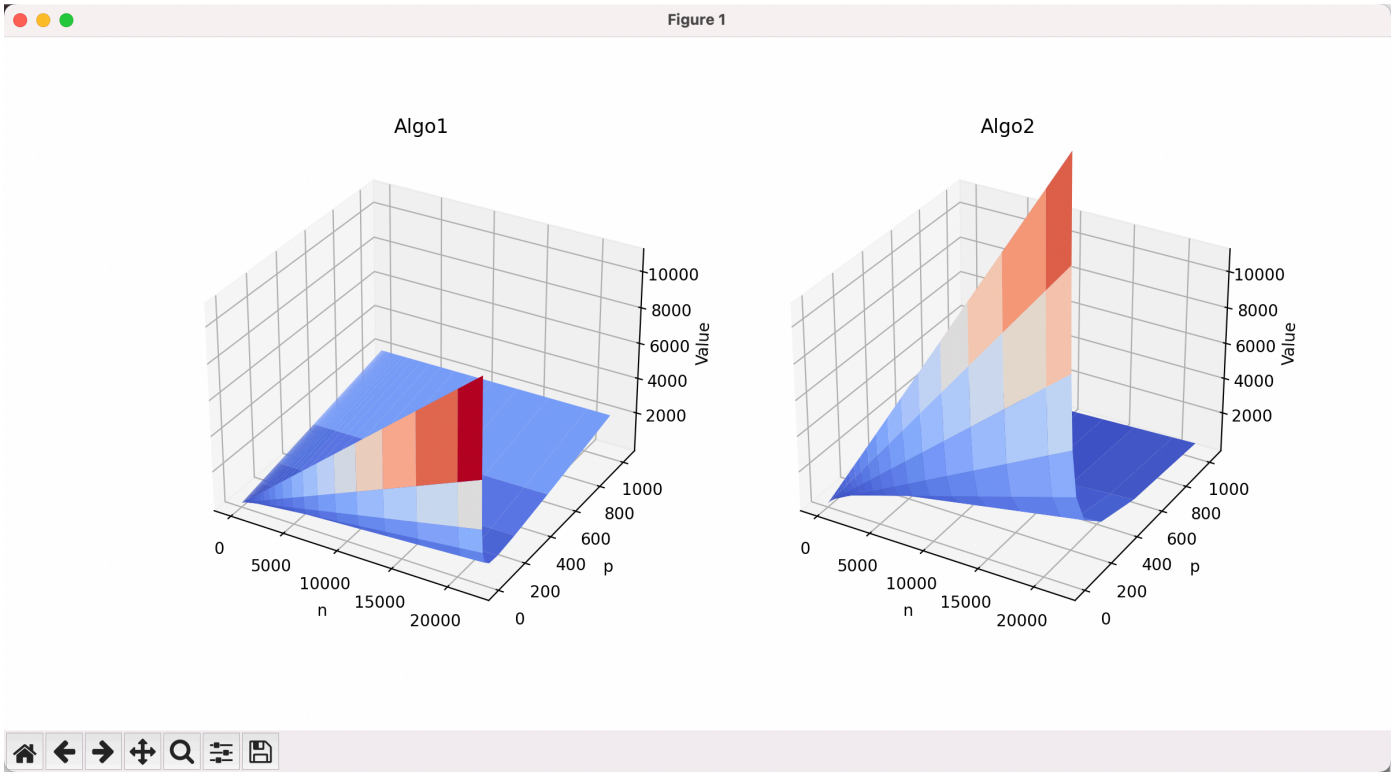
a. 在课本 1.3 节的例子中, 第一种计算全局总和的算法 (0 号核作为 master 核).

$n/p + 2p$

b. 在课本 1.3 节的例子中, 第二种计算全局总和的算法 (树形结构).

$2(n/p)\log p$

c. 制作一张表来比较这两种算法在总核数是 2、4、8、...、1024 时, 0 号核执行的接收与加法操作的次数.



2.2 习题 6

回顾之前一个从缓存读取二维数组的示例 (课本 2.2.3 的示例). 请问一个更大矩阵和一个更大的缓存是如何影响两对嵌套循环的性能的?

当二维数组变得更大时:

- 访问的数组元素数量增加, 这可能导致更多的缓存失效, 因为缓存的容量是有限的, 无法存储整个数组.
- 如果矩阵的大小远超过缓存容量, 那么缓存将无法有效地缓存数据, 导致每次迭代都可能需要从主存中加载数据, 这将显著降低性能.

当缓存变得更大时:

- 缓存可以存储更多的数组元素, 这减少了循环迭代中缓存失效的次数.
- 更少的缓存失效意味着 CPU 可以更频繁地从缓存中读取数据, 而不是从较慢的主存中读取, 从而提高了性能.

对于两对嵌套循环:

1. 第一对嵌套循环 (例如, 按行遍历):

- 如果缓存足够大, 能够存储至少一整行的数据, 那么按行遍历的性能通常较好, 因为缓存可以一次性加载整行数据, 并在后续列访问中重用这些数据.
- 如果缓存不够大, 无法存储一整行数据, 那么每次访问新列时都可能发生缓存失效, 导致性能下降.

2. 第二对嵌套循环 (例如, 按列遍历):
   - 按列遍历通常更容易受到缓存大小的影响, 因为每次迭代都可能访问不同的行. 如果缓存不够大, 无法存储足够多的行来覆盖整个列遍历过程, 那么性能会显著下降.
   - 较大的缓存可能能够存储更多的行数据, 减少按列遍历时的缓存失效次数, 从而提高性能.

如果 MAX = 8, 缓存可以存储 4 个缓存行, 情况又会是怎样的?

第一对嵌套循环发生 16 次, 第二对嵌套循环发生 64 次

在第一对嵌套循环中对 A 的读操作, 会导致发生多少次失效?

16 次

第二对嵌套循环中的失效次数又是多少?

64 次