并行计算介绍与算法设计

清华大学 韩文弢

2021年2月4日

课程提纲

- 并行计算基础介绍
- 并行计算的模型和实现
- 并行算法设计与例题
- 并行计算与分布式计算

并行计算基础介绍

引子：编程比赛的类型

- 主流编程比赛：OI系列、ICPC系列、Google Code Jam等商业比赛
  - 主要考察算法设计
  - 问题规模较小，题意独立（自包含）
  - 黑箱自动评测
- 缺乏
  - 大规模系统设计
  - 代码可维护性、可扩展性
  - 计算机科学其他领域的知识
- 其他类型的竞赛
  - CCSP（算法+系统）、Distributed GCJ（并行/分布式）、超算比赛（综合）

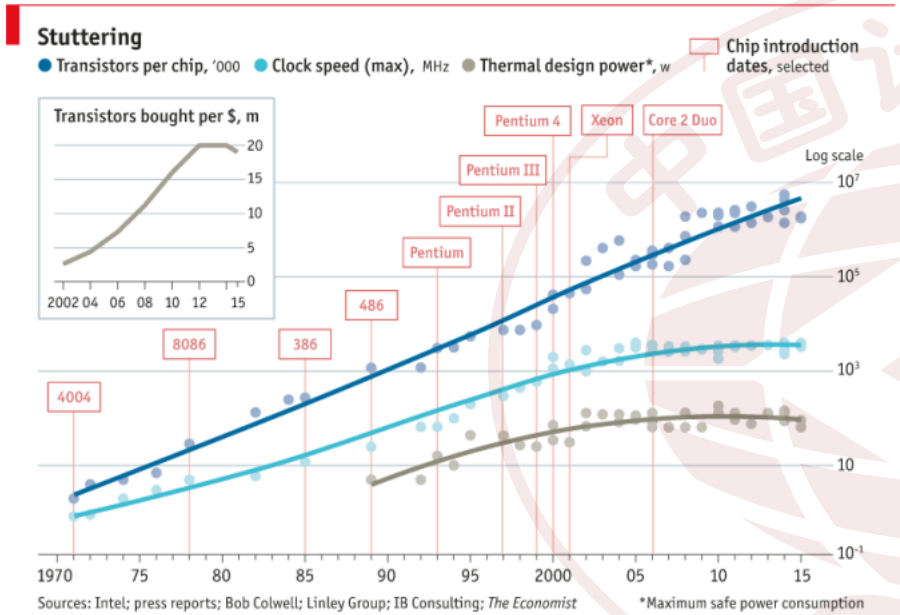
什么是并行计算？

- 串行处理：多个任务依次（轮流）执行

- 并行处理：多个任务同时处理

- 并行计算：一般是指许多指令得以同时进行的计算模式

为什么要用并行计算？



摩尔定律现状：晶体管集成度还在继续提升。然而时钟频率已经到达极限（功耗、延迟）。

时钟频率→单个运算设备的速度
集成度→单位面积运算设备的数量
总体运算速度=单个运算设备的速度×运算设备的数量

TOP500：超级计算机排名

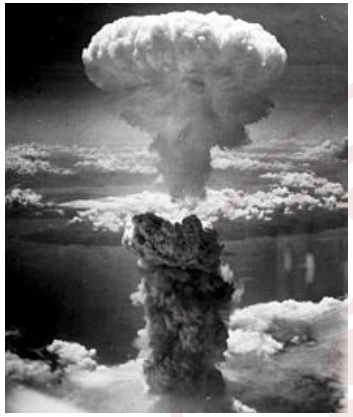
Rank	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442,010.0	537,212.0	29,899
2	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148,600.0	200,794.9	10,096
3	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94,640.0	125,712.0	7,438
4	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10,649,600	93,014.6	125,435.9	15,371
5	**Selene** - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555,520	63,460.0	79,215.0	2,646
6	**Tianhe-2A** - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4,981,760	61,444.5	100,678.7	18,482

- 核数众多（百万起）
- 每秒计算能力在百PFLOPS级别
  - 1PFLOPS=每秒10^{15}次浮点运算
- 功率很高，MW级别
- 中国、美国、日本、欧洲是主要玩家

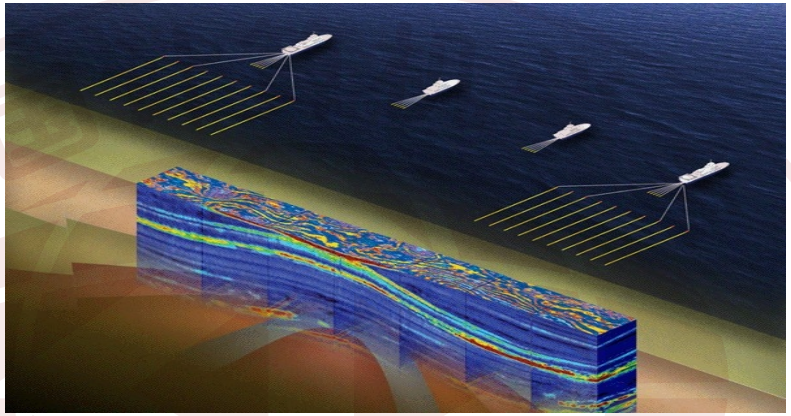
2020年11月

超算的用途

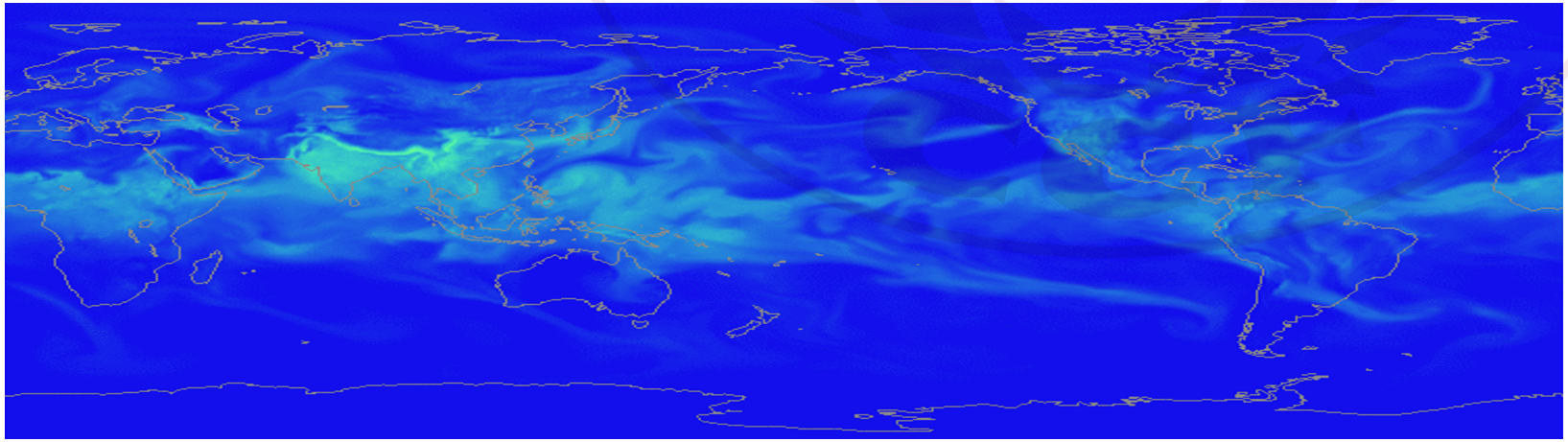
弹道计算



核爆模拟



地质勘探



气候模式/天气预报

万物皆可计算

- 物理
- 化学
- 生物
- 环境
- 社会
- ……

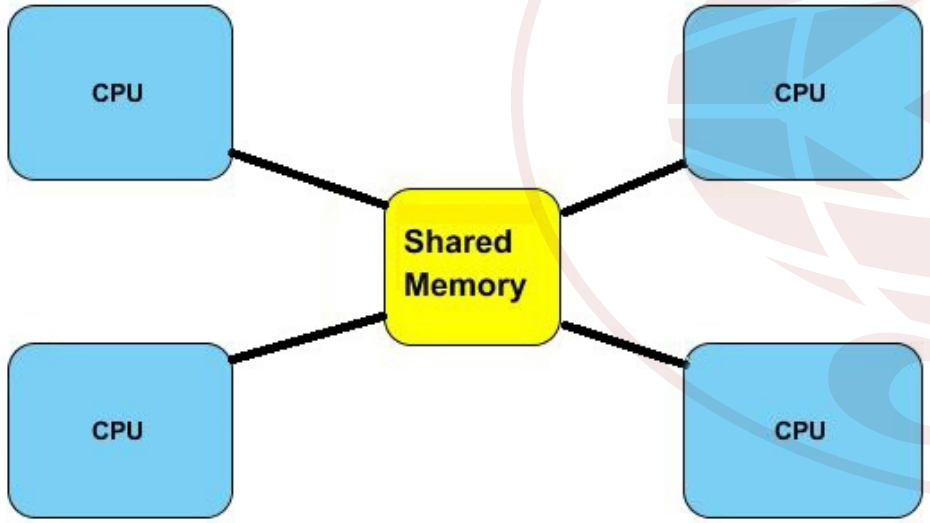
超算应用构建过程

1. 分析原理
2. 建立模型
3. 模拟计算

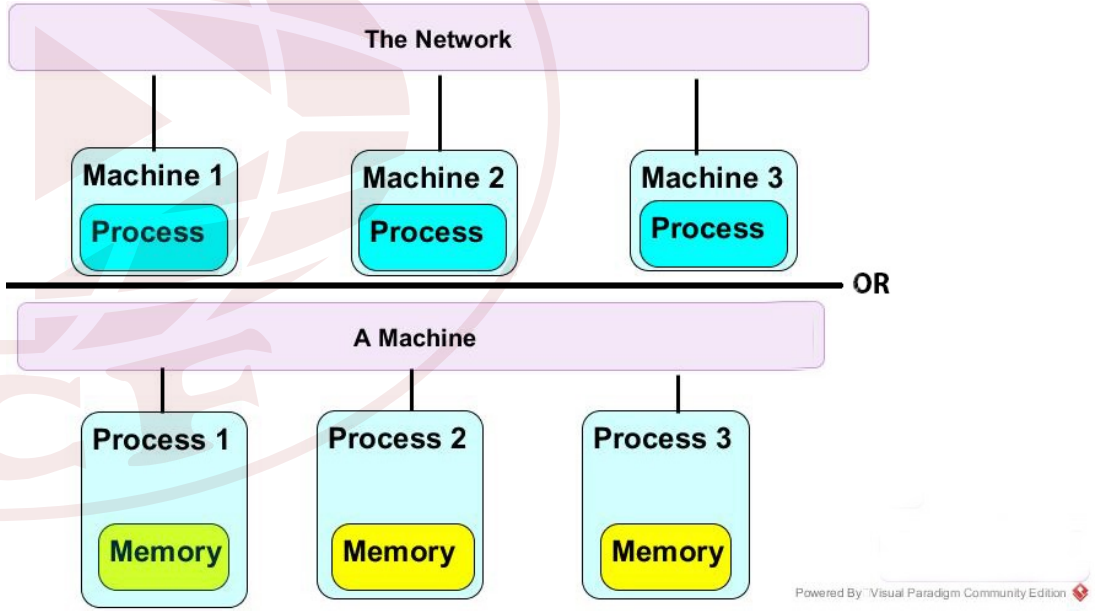
并行计算的模型和实现

并行计算的常见模型

共享内存 Shared Memory

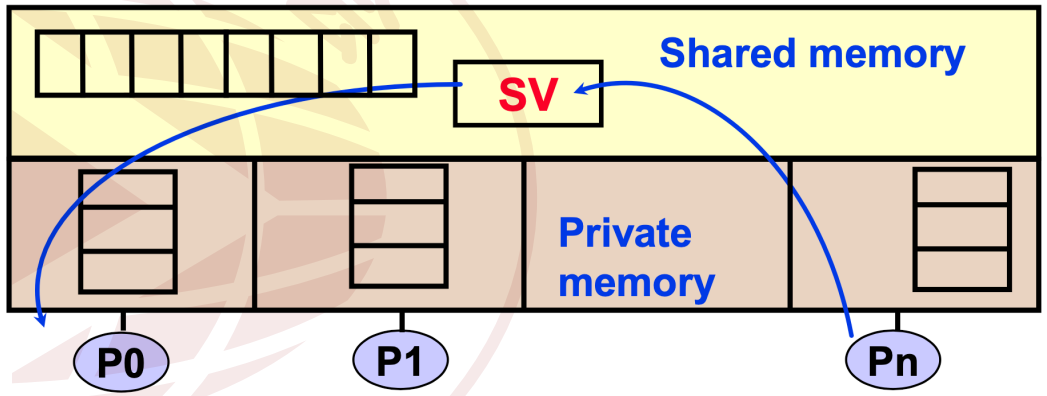


消息传递 Message Passing



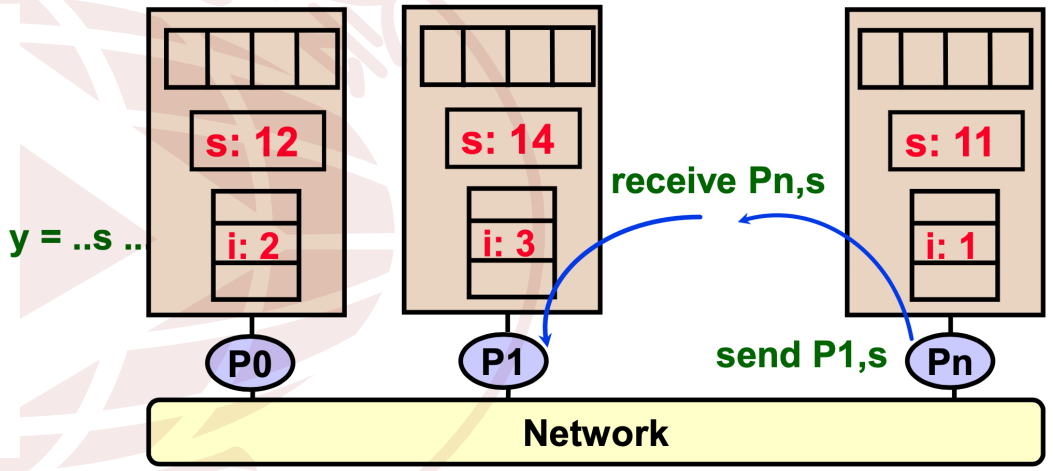
共享内存并行模型

- 程序由一组进程/线程组成
- 单一地址空间
- 局部变量和共享变量
- 通信由读写共享变量隐式完成
- 进程/线程异步执行
- 显式同步控制
- 线程可以动态产生和消亡
- 适用于单机多核并行的情况



消息传递并行模型

- 程序由一组命名的进程组成
- 分开的地址空间
- 没有共享数据
- 通信通过显式的发送/接收指令
- 进程异步执行
- 显式同步控制
- 进程数一般在程序启动时确定
- 适用于多机并行的情况



共享内存编程工具：Pthreads

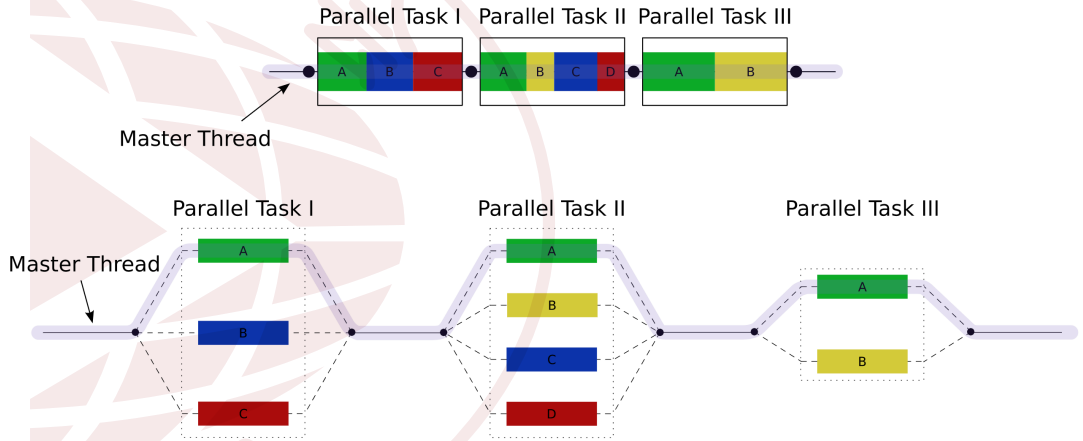
- 单机并行库，C语言接口
- 提供4组函数：
  - 线程管理
  - 互斥锁
  - 条件变量
  - 同步机制

```
#include <pthread.h>

void *worker(void *args) {
  ...
}

int main() {
  pthread_t tid[N];
  int args[N];

  for (int i = 0; i < N; i++) {
    args[i] = i;
    pthread_create(&tid[i], NULL, worker, &args[i]);
  }

  for (int i = 0; i < N; i++) {
    pthread_join(tid[i], NULL);
  }
}
```

共享内存编程工具：OpenMP

- 基于fork-join的多线程执行模型
- 以编译命令的方式来描述并行
- 更多并行调度等的支持

```
int main() {
  int a[100000];
  #pragma omp parallel for
  for (int i = 0; i < 100000; i++) {
    a[i] = 2 * i;
  }
  return 0;
}
```



共享内存编程工具：C++标准库

- 从C++11以来，C++标准库中加入了一系列并行编程的支持
  - 线程库 C++11
    - <thread>
    - <mutex>
    - <condition_variable>
    - …
  - 并行算法 C++17
    - <algorithm>

并行算法举例：

```
template< class ExecutionPolicy, class Forwa
rdIt, class UnaryFunction2 >
void for_each( ExecutionPolicy&& policy,
ForwardIt first, ForwardIt last,
UnaryFunction2 f );
```

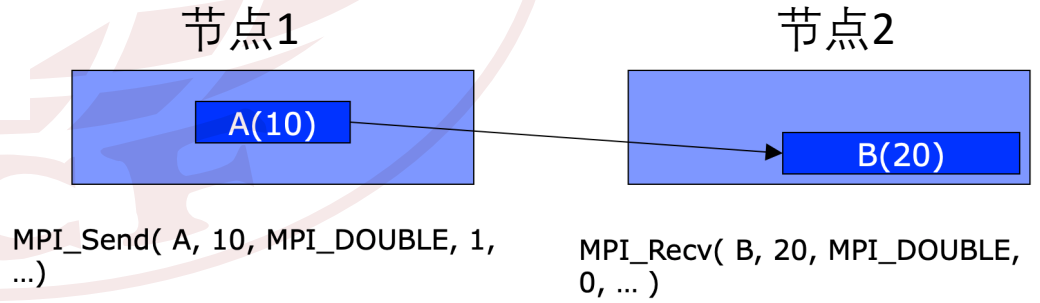
并行策略：seq，par

```
std::for_each(std::execution::par, a.begin(),
                    a.end(), [](auto& x) {...});
```

消息传递编程工具：MPI

- 消息传递并行编程的API
  - MPI_Init
  - MPI_Finalize
  - MPI_Comm_size
  - MPI_Comm_rank
  - MPI_Send
  - MPI_Recv
  - …
- 是超算并行计算的主流框架
- MPI+OpenMP混合

发送/接收接口：
MPI_Send(start, count, datatype, dest, tag, comm)
MPI_Recv(start, count, datatype, source, tag, comm, status)



并行算法设计与例题

并行算法比赛介绍

- 与IOI类似，提交单个程序，由框架编译并执行
- 每个节点都运行这个相同的程序
- 节点之间可以进行通信，完成相互协作
  - 一个简化版MPI的消息传递协议
- 例子：Distributed Google Code Jam (2015-2018)

- int NumberOfNodes()
- int MyNodeId()
- PutInt(int target, int value)
- Send(int target)
- Receive(int source)
- GetInt(int source)

例题1：约数 Divisors

给定一个正整数$n \leq 10^{18}$，统计它的约数个数。

串行解：

从1到$\sqrt{n}$依次枚举d，如果n能够被d整除，则计数2（$d^2 \neq n$）或1（$d^2 = n$）。

时间复杂度为$O(\sqrt{n})$。

```
int main() {
  long long n;
  int num_divisors = 0;
  cin >> n;
  for (long long d = 1; d * d <= n;
       d++) {
    if (n % d == 0) {
      num_divisors++;
      if (d * d != n) num_divisors++;
    }
  }
  cout << num_divisors << endl;
}
```

例题1：约数 Divisors

并行解：

使用M个节点进行并行计算。

- 如何划分任务？
  - 节点0负责$1, 1+M, 1+2M$ …
  - 节点1负责$2, 2+M, 2+2M$ …
  - ……
- 如何计算任务？
  - 与串行相同
- 如何得到最终结果？
  - 求和

计算时间复杂度为$O(\sqrt{n}/M)$。

其他任务划分方案？

```
...
for (long long d = MyNodeId() + 1; d * d <= n;
     d += NumberOfNodes()) {
  if (n % d == 0) {
    num_divisors++;
    if (d * d != n) num_divisors++;
  }
}
if (MyNodeId() > 0) {
  PutInt(0, num_divisors);
  Send(0);
} else {  // MyNodeId() == 0
  for (int i = 1; i < NumberOfNodes(); i++) {
    Receive(i);
    num_divisors += GetInt(i);
  }
}
...
```

通信的复杂度？

例题2：工坊 Workshop

一群学生围成圆圈想问题。当一个学生想到解法时，他可以和他相邻的两个人分享，这个过程要用一分钟时间。获得解法的学生可以继续跟他相邻的人分享（也是用一分钟时间），以此类推。

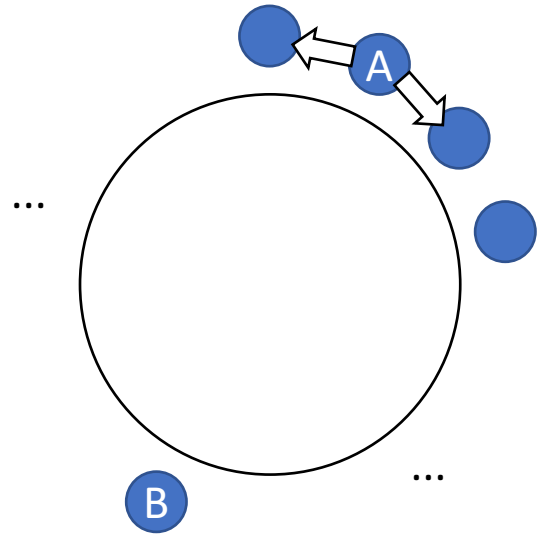
如果学生A想到了解法，问学生B在多久之后能够学到这个解法（假设除学生A外其他人都是通过分享学到解法的）。会有多组这样的查询。

输入：

由6个交互库函数给出输入数据（见下页）。

输出：

每行一个查询的结果，表示分享解法所需的时间。



例题2：工坊 Workshop

- `int NumberOfStudents()`：返回参加工坊的学生人数n，学生从1到n连续编号（$3 \le n \le 10^9$）。
- `int FirstNeighbor(int i)`：返回第i个学生的第一个邻居的编号（所谓第一个邻居是指两个邻居中编号较小的那个）。
- `int SecondNeighbor(int i)`：返回第 i 个学生的第二个邻居的编号。
- `int NumberOfQueries()`：返回查询个数m，查询从1到m连续编号（$1 \le m \le 200$）。
- `int QueryFrom(int i)`：对于第i个查询，返回在这个查询中想到解法的那个学生的编号。
- `int QueryTo(int i)`：对于第i个查询，返回想要知道解法的那个学生的编号。

例题2：工坊 Workshop

题意分析：

本题没有直接给出圆圈上学生的排布，而且通过一个交互函数获得每个学生两侧的邻居，但是没有明确左右关系。

学生在圆圈上的排布是固定的，每次查询都是一个独立的场景。也就是求圆圈上某两个学生之间的最短距离。

串行解：

从要查询的一个学生开始，不断通过函数调用获得邻居，确定沿两个方向的排布，直到发现查询的另一个学生。时间复杂度为 $O(n)$ 。

例题2：工坊 Workshop

并行解：

从一些点开始遍历，得到一个“邻域”的情况，然后把这些“邻域”情况拼起来得到全局的最短距离。

- 如何划分任务？
  - 取一些点作为“检查点”（如何取点？），将这些检查点平均分配给计算节点
- 如何计算任务？
  - 对于每个节点，依次处理分配到的检查点，用串行解类似的办法向两边遍历，直到遇到最近的另外一个检查点
- 如何得到最终结果？
  - 计算任务后将所有两个检查点的编号和他们之间的距离信息发送给某个节点汇总，得到最终答案

例题2：工坊 Workshop

取点方案：

- 待查询的点（不超过$2m$个）
  - 数量不一定够分
  - 存在退化情况（并行算法的效率取决于最慢的计算节点）
- 再选择一些随机编号的点
  - 数量为$O(M)$，其中M是计算节点个数
  - 用确定性的伪随机函数生成
  - 可以证明最慢的节点上的复杂度为$O(\frac{n \log M}{M})$
- 总的时间复杂度？

例题3：助手 Assistant

助手要修订一篇文章，可以使用的编辑操作方式有：插入一个字符、修改一个字符、删除一个字符。他希望编辑操作数越少越好。另外，他还倾向于字母序较小的字母，希望在编辑操作数最少的前提下，把字母改“大”的操作数最少。

输入：

- 第一行包含两个整数k和l（$1 \leq k, l \leq 100000$），表示修订前后文章的长度。
- 随后两行是两个字符串，分别是修订前后文章的内容，仅由小写字母组成。

输出：

两个整数，分别是最少的编辑操作数，以及满足前面条件下最少的把字母改“大”的操作数。

例题3：助手 Assistant

串行解：

编辑距离问题的扩展版本，使用动态规划思想，将编辑操作数和字母改大的操作数一起作为状态来考虑，算法的时间复杂度为$O(kl)$。

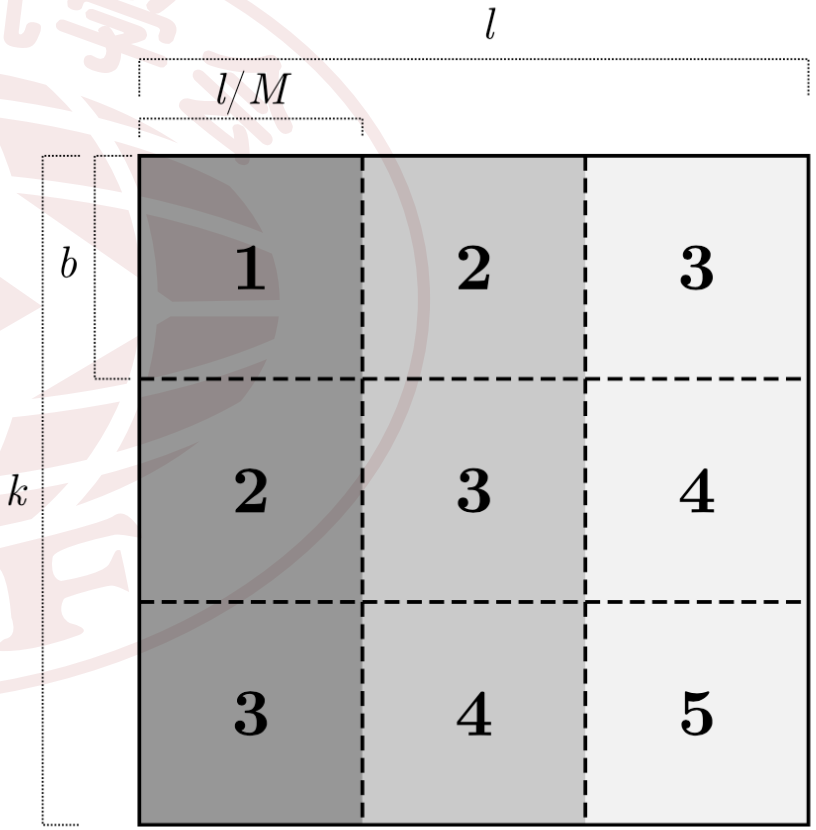
并行解：

- 如何划分任务？
  - 动态规划过程在一个矩阵中进行，对矩阵进行合理的划分，使得计算可以并行进行
- 如何计算任务？
  - 按串行方法进行计算，需要注意状态转移时依赖的状态可能在其他计算节点上，需要通过通信获取
- 如何得到最终结果？
  - 直接获取

例题3：助手 Assistant

划分方法：

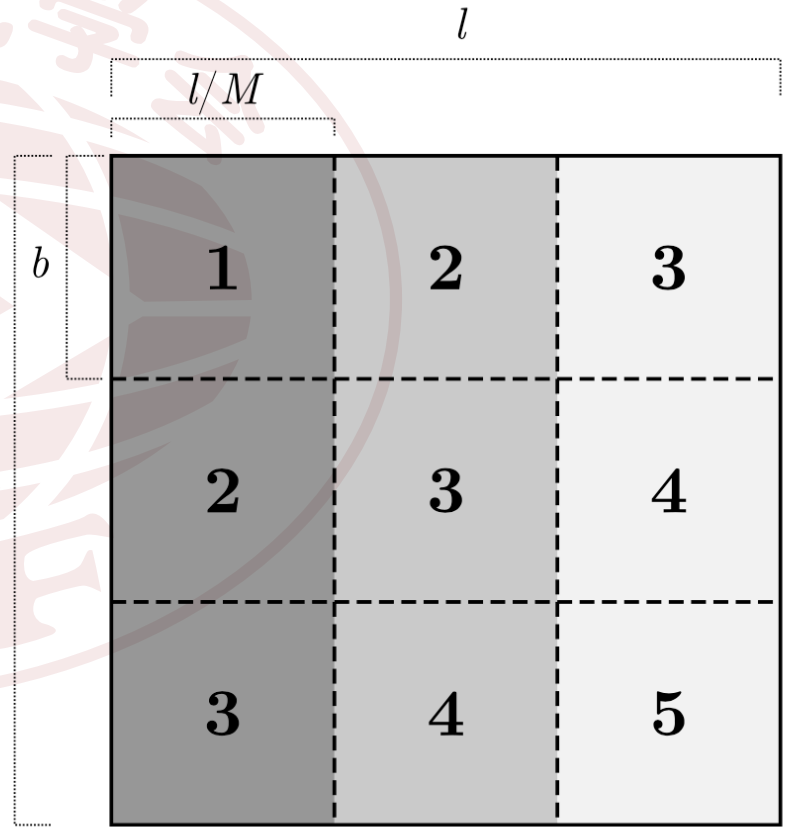
- 设矩阵的行代表修订前的字符串，列代表修订后的字符串
- 将矩阵按列分M块，每块分配给一个节点
  - 如果一个列块计算完成之后再算下一个列块，没有并行效果
  - 如果一个列块中每一行算完就把结果发给下一个列块，通信次数为$O(kM)$
- 怎么办？折中！



例题3：助手 Assistant

划分方法：

- 按行也分块，设每块高度为b
- 每一小块计算完成后向右通信
- 最慢节点的等待时间为$O(bl)$，计算时间为$O(kl/M)$，总的时间复杂度为$O(bl + kl/M)$
- 取$b = k/M$，则时间复杂度为$O(kl/M)$
- 消息数量为$O(M^2)$。通信量？



并行算法设计方法

- 关键：如何划分任务？
  - 尽量平均（负载均衡）
  - 降低依赖
  - 减少通信
- 与分治算法比较
- 最优解
- 可扩展性

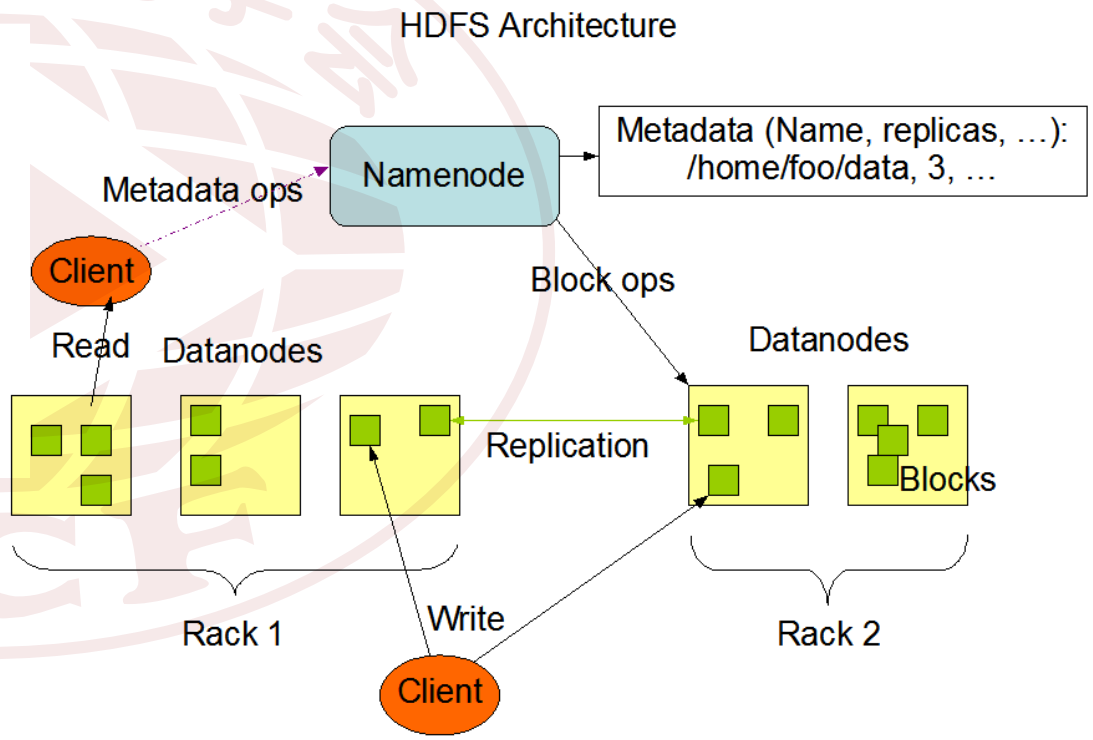
并行计算与分布式计算

分布式计算

- 主要场景：在数据中心处理大数据
- 常见系统：Hadoop、Spark
- 设计要点
  - 普通机器
  - 长时间运行
  - 容错
- 提供服务
  - 存储，如GFS、HDFS
  - 计算，如MapReduce

分布式文件系统HDFS

- 存储大量大文件
- 设计要点
  - 主-从架构
  - 文件分块
  - 数据块多副本
- 系统组成
  - 元数据服务器 Namenode
  - 数据服务器 Datanode
  - 客户端 Client



分布式计算框架MapReduce

- 模型
  - 数据以键值对为单元
  - 由map和reduce两个阶段组成
- 过程
  - map：根据原始数据生成(key, value)键值对
  - 汇聚：将相同key的键值对汇聚到一起
  - reduce：对相同key的值进行归约，得到新的键值对
- 例子：统计文本中单词的出现频率
- map：对于一个单词 w，生成(w, 1)
- reduce：对于 (w, 1的数组)，统计1的个数n，生成(w,n)

并行计算vs分布式计算

- 相同点
  - 多于一个执行单元
  - 有信息交换途径
  - 能够提高效率
- 不同点
  - 应用场景
  - 是否跨机器
  - 信息交换方式
  - 故障处理方式

课程小结

- 并行计算是目前超算的主要实现方式
- 并行计算有共享内存和消息传递两种主要形式
- 并行算法设计在串行算法设计的基础上进一步考虑任务划分、通信模式等因素
- 并行计算与分布式计算都是多个处理单元同时工作，各有侧重