

STAT243 Lecture 6.2 Overview of parallel processing

1 计算机体系结构 (Computer Architecture)

- 多处理器趋势
 - 由于物理极限，单个处理器的速度提升变得困难，芯片行业转向在同一计算机上集成多个处理单元
 - 多处理器 (multi-processor) 和多核心 (multi-core) 架构因此成为主流
- 多核心计算机 (Multi-core computers)
 - 每个 CPU 通常包含多个核心 (core)
 - 在个人电脑中，多个处理器或多核心共享同一主存 (shared memory)
- 超级计算机与集群 (Supercomputers and clusters)
 - 由大量节点 (nodes) 组成，每个节点拥有独立的 CPU 与内存
 - 节点间通过高速网络连接，采用 分布式内存 (distributed memory) 结构
 - 内存访问原则：访问本地内存远快于跨节点通信

Example

示例：Perlmutter 超级计算机 (Lawrence Berkeley Lab)

- 3072 个 CPU-only 节点 + 1792 个 GPU 节点
- 总约 50 万 CPU 核心
- 每节点 512 GB 内存，总计约 2.3 PB

Logic

在学习中可不区分多核心与多处理器，重点在于是否 共享内存

2 常用术语 (Terminology)

术语	含义
cores	单台机器或单节点中的多个处理单元，例如 AMD EPYC 7763 每个 CPU 含 64 核心
nodes	集群中的独立计算机，每个节点有自己的内存
processes	程序的执行实例，可并发运行，理想情况下进程数不超过核心数
workers	实际执行并行任务的进程，与 process 可互换使用
tasks	单个计算单元，每个 process 在某个 core 上执行一个或多个 task
threads	单个进程中的多执行路径，被视为轻量级进程
forking	创建子进程，拥有独立的内存与进程 ID；未修改的对象可能共享父进程内存
scheduler	管理集群任务的调度程序，例如 Slurm
load-balanced	所有核心在计算过程中都被充分利用
sockets	R 中通过 socket 通信实现并行 (如多个 Rscript 进程间的交互)

3 共享内存与分布式内存 (Shared vs Distributed Memory)

类型	结构	特点
共享内存 (Shared Memory)	多核心共享同一内存空间	无需显式消息传递，但新建进程时可能复制对象
分布式内存 (Distributed Memory)	各节点独立内存，通过网络连接	依赖消息传递 (Message Passing) 机制如 MPI

3.1 共享内存 (Shared Memory)

- 所有核心访问相同内存，无需显式通信
- 使用线程 (threading) 时可让多个线程访问同一对象而不复制，否则创建新进程时仍会复制对象
- Python 与 R 通常能自动防止不同线程写入同一内存位置

🔗 Logic ▾

本单元将主要介绍两种共享内存并行方式

- 线程化线性代数 (Threaded Linear Algebra)
- 多核心功能 (Multicore Functionality)

3.1.1 Threading

- Threads 是单进程内的多执行路径
- 在系统监控工具 (如 Linux/macOS 的 `top`) 中，threaded code 可能使用超过 100% CPU，代表多核并行

⚠ Remark ▾

与 超线程 (Hyperthreading) 不同：超线程让一个核心在操作系统层面表现为两个逻辑核心

3.2 分布式内存 (Distributed Memory)

- 节点间通信需通过消息传递实现
- 标准协议为 MPI (Message Passing Interface)，常见实现包括 openMPI
- Python 与 R 的一些包在底层使用 MPI

🔗 Logic ▾

本课程聚焦使用 Dask 进行分布式并行 (非 MPI 实现)

4 GPU 加速 (GPUs)

- GPU (图形处理单元) 最初用于图像渲染，具有大量简单计算单元，可进行 大规模并行计算 (massively parallel computation)
- GPGPU (General-Purpose GPU Computing) 将 GPU 用于通用计算
- 使用方式：
 - 多数研究者不直接编写 GPU 程序，而使用能自动利用 GPU 的软件，例如 TensorFlow, PyTorch, JAX
 - GPU 执行的函数是在 GPU kernel 中执行的，CPU 负责总体流程，将计算密集部分交由 GPU

- GPU 与 CPU 主存独立，应避免频繁数据传输
 - 启动 kernel 有额外开销，应避免过多微小任务
 - GPU 与 TPU 等协处理单元常被称为 **co-processors**，用于辅助 CPU 计算
-

| 5 其他并行处理方式 (Other Parallel Approaches)

| 5.1 Spark 与 Hadoop

- 在 **分布式内存环境** 中实现计算
- 使用 **MapReduce 框架** 进行数据处理 (详见 Unit 7)

| 5.2 云计算 (Cloud Computing)

- 云服务提供商: **AWS (EC2)**、**Google Cloud (GCP)**、**Microsoft Azure**
- 采用 **按需付费 (pay-as-you-go)** 模式，租用虚拟机 (VM)
- 用户可选择操作系统 (Linux/Windows)、核心数量、内存配置
- 虚拟机可像物理机一样安装应用、加载数据、远程登录
- 可组合多台虚拟机形成 **虚拟集群 (virtual cluster)**，在云端运行数据库或 Spark 等平台