16 性能优化 （局部性，通信和竞争）

- 消息传递
  - 同步（阻塞式的）
  - 异步（非阻塞式的）
- **延迟和吞吐**
  - 传输延迟(Latency): 一个操作完成需要的时间.
  - 吞吐量(Throughput/Bandwidth): 操作执行时的速度
    - 加快传输速度
    - 加大带宽
- 流水线——提高吞吐但不提高延迟
  - 非流水线

Example: sending a n-bit message

$$T(n) = T_0 + \frac{n}{B}$$

$T(n)$ = transfer time (overall latency of the operation)

T_0 = start-up latency (e.g., time until first bit arrives at destination)

n = bytes transferred in operation

B = transfer rate (bandwidth of the link)

  - 通信开销
    - 通信时间=本地开销+传输开销+传播开销+异地开销
    - 通信开销=通信时间-overlap 部分（通信和计算的重叠）
- **通信**
  - 天然通信
    - 基本上必须在处理器之间移动以执行给定分配的算法的信息（假设无限容量缓存、最小粒度传输等）
    - 并行算法中必须要的通信, 是算法的组成部分
    - 减少天然通信
      - 好的划分方式：增加运算密度，改进算法
      - 例子：PPT27一维块划分---->二维块划分
  - 人为通信
    - 所有其他沟通（人工沟通源于系统实施的实际细节）
    - 例子：
      - 系统可能具有最小的传输粒度（结果：系统必须传递比所需更多的数据）
        - 程序加载一个 4 字节浮点值，但必须从内存中传输整个 64 字节高速缓存行（比必要的通信量多 16 倍）
      - 系统可能具有导致不必要通信的操作规则：

        - 程序存储 16 个连续的 4 字节浮点值，但整个 64 字节缓存线首先从内存中加载，然后存储到内存中（2x 开销）
    - 分布式内存中的数据放置不当（数据不在访问最多的处理器附近）
    - 有限的复制容量（由于本地存储（例如缓存）太小而无法在访问之间保留它，因此多次与处理器通信相同的数据）
- **运算密度**: 运算数量/通信数量
- cache
    - cold miss
        - 第一次接触数据。 在顺序程序中不可避免
    - Capacity miss
        - 工作集大于缓存。 可以通过增加缓存大小来避免/减少。
    - Conflict miss
        - 缓存管理策略导致的缺失。 可以通过更改缓存关联性或应用程序中的数据访问模式来避免/减少。
    - Communication miss
        - 由于并行系统中固有的或人为的通信
- 减少通信的技术
    - 改变访存顺序
    - 合并循环
    - 共享数据增加计算密度
        - 利用共享：共同定位对相同数据进行操作的任务
            - 调度在同一处理器上同时处理同一数据结构的线程
            - 减少**天然通信**
        - 示例：CUDA 线程块
            - 用于在 CUDA 程序中本地化相关处理的抽象
            - 块中的线程通常协作执行操作（利用通过 CUDA 共享内存快速访问/同步）
            - 所以 GPU 实现总是从同一个 GPU 核心上的同一个块调度线程
    - 利用空间局部性
        - 沟通的粒度可能很重要，因为它可能会引入**人为通信**
        - 通信/数据传输的粒度
        - 缓存一致性的粒度
- 竞争
    - 资源可以在给定的吞吐量（每单位时间的事务数）下执行操作
        - 内存、通信链路、服务器等
    - 当在很短的时间内对资源发出许多请求时会发生竞争（资源是“热点”）
    - 减少竞争

        - 分布式工作队列可以减少争用
        - 在大型并行机器上创建粒子数据结构网格，在单元格上并行化
        - 粒子层次并行化
        - 使用细粒度锁——对链表的每个元素有锁
        - 计算部分结果 + 合并
- 减少通信开销
    - 减少与发送方/接收方的通信开销
        - 发送更少的消息，使消息更大（摊销开销）
        - Coalesce （合并）许多小消息变成大消息
    - 减少延迟
        - 应用程序编写者：重构代码以利用局部性
        - 硬件实现者：改进通信架构
    - 减少竞争
        - 复制竞争资源（例如，本地副本、细粒度锁）
        - 错开对竞争资源的访问
    - 增加通信/计算重叠
        - 应用程序编写器：使用异步通信（例如，异步消息）
        - 硬件实现者：流水线、多线程、预取、乱序执行
        - 在应用程序中需要额外的并发性（并发性多于执行单元的数量）
- 天然与人为通信
    - 考虑到问题是如何分解以及工作是如何分配的，天然通信是基础
    - 人为通信取决于机器实现细节（通常与天然通信一样重要）
- 提高程序性能
    - 识别和利用局部性：减少交流（增加算术强度）
    - 减少开销（更少的大型消息）
    - 减少竞争
    - 最大化通信和处理的重叠（隐藏延迟以免产生成本）

以上内容整理于 幕布文档