4 并行编程方法论

- 增量式并行化
  - **什么是增量式并行化**
    - 研究一个串行程序（或代码段）
    - 寻找并行的瓶颈和机会（串行程序中执行时间最长的部分）
    - 尽量让所有处理器忙于做有用的工作
- 并行化方法论
  - **Culler's 设计方法论** PPT16页有图
    - 分解
      - 将问题打包成可以并行执行的任务
      - 不需要静态执行，可以静态也可以静态分解
      - 可以在程序执行时识别新任务
      - 核心思想：创建最少任务且使得所有的机器上的执行单元都处于忙碌状态
      - 关键方面：识别出依赖部分。（identifying dependencies）
      - 分解是程序员的工作
      - 动态分解串行程序一直是一个挑战性问题:编译器必须分析程序，识别依赖……
    - 分配
      - 分发任务给线程
      - 目标：负载均衡，减少通信开销。
      - 特点：静态动态皆可，一般需要程序员来负责，也有非常多语言可以自动对此负责。
    - 配置Orchestration
      - 结构化通信（Structuring communication）
      - 增加同步来保证必要的依赖性（Adding synchronization to preserve dependencies if necessary）
      - 在内存中组织数据结构
      - 调度任务
      - 目的：减少通信和同步的开销，保护数据的局部性，减少额外开销(overhead).
      - 机器细节影响许多决定：如果同步是昂贵的，那就少用
    - 映射
      - 将线程映射到硬件执行单元上
      - 执行对象：OS、编译器、硬件
      - 一些有趣的映射决定

        - 在相同的处理器放置相关的/不相关（减少彼此的干扰，也许一个受限于访存另一个受限于计算）的线程
- **Foster's设计方法论**
    - Partitioning 分解
        - 将计算和数据分成几部分
        - 利用数据并行性：将数据分成几块（数据/域 划分/分解）；确定如何将计算与数据相关联
            - **按域或数据分解**
                - 首先，决定如何在处理器之间划分数据元素
                - 其次，决定每个处理器应该执行哪些任务
                - 示例：查找向量中的最大元素PPT29：先划分给不同CPU，CPU自己内部比，然后在不同CPU比较
        - 利用任务并行性：将计算分成几部分，（任务/功能划分/分解）确定如何将数据与计算关联
            - **任务或功能划分**
                - 首先，在处理器之间划分任务
                - 其次，决定哪个处理器将访问（读取和/或写入）哪些数据元素
                - 示例：GUI 的事件处理程序
        - 利用流水线并行性（优化循环）
            - 特殊类型的任务分解
            - “装配线”平行度
            - 示例：计算机图形中的 3D 渲染
            - 流水线加速比
                - 流水线提升吞吐量但没有改善延迟
        - **分解要注意的问题**
            - 至少比目标计算机中的处理器多 10 倍的原始任务。如果没有，以后的设计选项可能会受到太多限制
            - 最小化冗余计算和冗余数据存储，否则，当问题规模增大时，设计可能无法正常工作
            - 原始任务大小大致相同，如果没有，可能很难平衡处理器之间的工作
            - 任务数是问题大小的增函数，如果不是，可能无法使用更多处理器来解决大问题实例
    - Communication 通信
        - 确定任务之间传递的值——任务通道图
        - 本地通信
            - 任务需要来自少量其他任务的值
            - 创建说明数据流的通道

- 全局通信
  - 大量任务贡献数据来执行计算
  - 不要在设计早期为他们创建渠道
- 通信注意的问题
  - 通信是并行算法的开销，我们需要将其最小化
  - 任务之间的通信操作平衡
  - 每个任务只与一小部分邻居通信
  - 任务可以同时执行通信
  - 任务可以同时执行计算

- Agglomeration (归并、组合)
  - 将任务组成更大的任务
  - 目标
    - 提升性能
    - 保持程序的可扩展性
    - 简化编程（降低软件工程成本）
  - 在消息传递编程中，目标通常是为每个处理器创建一个聚合任务
  - **整合的意义**
    - 提升性能
      - 聚集成统一任务，消除原始任务之间的通信
      - 组合发送和接收任务组
    - 保持程序的可扩展性
      - 假设我们要开发一个并行程序来操作一个大小为 8 x 128 x 256 的 3D 矩阵。
      - 如果我们聚合第二维和第三维，我们将无法将程序移植到具有超过 8 个 CPU 的并行计算机上
    - 减少程序源代码量
      - 如果我们正在并行化一个串行程序，一个聚合可以让我们更多地利用现有的顺串行代码，减少开发并行程序的时间和费用。
  - 整合要满足的问题
    - 并行算法的局部性增加
    - 复制的计算比它们所取代的通信花费更少的时间
    - 数据拷贝不影响可扩展性
    - 聚集任务具有相似的计算和通信开销——均衡
    - 任务数量随着问题规模的增加而增加
    - 任务数量与硬件系统适配
    - 整合和代码修改成本之间的权衡是合理的

- Mapping 映射
  - 把任务分配到处理器上的过程
    - 集中式多处理器：由操作系统完成的映射
    - 分布式内存系统：由用户完成的映射
  - 映射的冲突目标
    - 最大化处理器利用率-----分散到不同处理器上的任务数就多----通信多
    - 最小化处理器间通信
  - **映射决策树**
    - 静态任务数
      - 结构化通信——可用几何图形表达，消息传递
        - 每个任务的恒定计算时间
          - 聚合任务以最小化通信
          - 每个处理器创建一个任务
        - 每个任务的可变计算时间——循环地将任务映射到处理器
      - 非结构化通信——使用静态负载平衡算法
    - 动态任务数
      - 任务之间的频繁通信——使用动态负载平衡算法
      - 许多生命周期短任务——使用运行时任务调度算法
  - 映射应该注意的问题
    - 基于每个处理器一个任务和每个处理器多个任务的考虑设计
    - 评估静态和动态任务分配
    - 如果选择动态任务分配，任务分配器不会成为性能瓶颈
    - 如果选择静态任务分配，任务与处理器的比例至少为 10:1
  - **并行设计举例**——边界值-散热问题
- 依赖图——有向图PPT91
  - 每个节点…
    - 变量赋值（索引变量除外）
    - 持续的
    - 运算符或函数调用
  - 边表示数据/控制相关性
    - 数据流（a=b）：变量的新值依赖于另一个值
    - 控制流（if）：在计算条件之前无法计算变量的新值
  - 更多并行化的探索
    - 三种基本并行化:数据、任务、流水线
    - 他们中的某些可以被结合