1 并行计算概览

- 并行与分布式系统
  - 分布式系统与并行系统交集：MPI/MapReduce
  - 并行系统与集中系统交集：OpenMP/TBB
- 并行计算总览
  - 并行架构、算法、编程、程序性能、应用
- **什么是并行计算?**
  - 同时利用多个计算资源解决一个计算问题
    - 程序运行在多个CPU上
    - 一个问题被分解为离散可并发解决的小部分
    - 每一小部分被进一步分解为一组指令序列
    - 每一部分的指令在不同CPU上同时执行
    - 需要一个全局的控制和协调机制
  - 可并行的计算问题应该满足
    - 可以分解为同时计算的几个离散片段
    - 在任意时刻可以执行多条指令
    - 多计算资源搜花的时间少于但计算资源的
  - 计算资源
    - 多处理器/多核的单台主机
    - 通过网络连接的若干数量主机
- **并行计算的优势**
  - 在自然界，很多复杂的、交叉发生的事件是同时发生的，但是又在同一个时间序列中；
  - 与串行计算相比，并行计算更擅长建模、模拟、理解真实复杂的现象
  - 节省时间和花销
    - 理论上，给一个任务投入更多的资源将缩短任务的完成时间，减少潜在的代价；
    - 并行计算机可以由多个便宜、通用计算资源构成；
  - 解决更大或更复杂的问题
    - 很多问题很复杂，不实际也不可能在单台计算机上解决，例如：Grand Challenges
  - 实现并发处理
    - 单台计算机只能做一件事情，而多台计算机却可以同时做几件事情
    - 例如协作网络，来自世界各地的人可以同时工作
  - 利用非本地资源

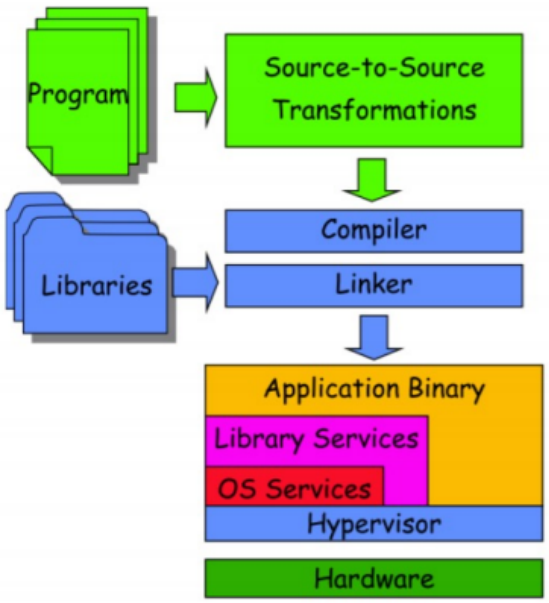
        - 当本地计算资源稀缺或者不充足时，可以利用甚至是来自互联网的计算资源。
    - 更好地发挥底层并行硬件
        - 现代计算机甚至笔记本都具有多个处理器或者核心；
        - 并行软件就是为了针对并行硬件架构出现的；
        - 串行程序运行在现代计算机上会浪费计算资源；
- **并行计算的用途**
    - 科学和工程计算
        - 大气、地球、环境；物理核能、例子模拟、高压、高分子；生物科技、遗传学
    - 工业和商业用途
        - 大数据、数据库、数据挖掘；石油勘探；Web搜索引擎、医学图像处理
    - 在全球范围各个领域得到广泛应用
- **并行计算发展的驱动力**
    - 应用发展趋势
        - 在硬件可达到的性能与应用对性能的需求之间存在正反馈
        - 大量设备、用户、内容涌现
        - 大数据：大数据正逐渐成为新的生产资料
        - 云计算兴起：廉价硬件，应用弹性扩展；应用种类繁多，负载异构型增加
    - 架构发展趋势
        - CPU架构技术经历了四代：电子管、晶体管、集成电路、大规模集成电路（VLSI）

          VLSI最的特色是在于对并行化的利用，不同的VLSI时代具有不同的并行粒度：
          1). 1985年之前：bit水平的并行，从4bit->8 bit->16bit （数据通路）
              a. 32bit后并行化速度放慢?
              b. 最近几年64bit才被广泛使用，128bit还很遥远，（非性能限制）
              c. 32bit并行技术的应用导致了计算机性能的显著提高
          2). 80年代到90年代：指令水平的并行
              a. 流水线、简单指令集、先进的编译技术（RISC）
              b. 片上缓存（caches）和功能部件 =>超变量执行
              c. 更复杂控制机制：乱序执行、推测、预测，用于控制转换和延迟问题；
          3).线程水平的并行
              现代并行计算的主要方式：线程并行。

        - VLSI 最的特色是在于对并行化的利用，不同的 VLSI 时代具有不同的并行粒度：
            - bit 级并行
            - 指令级并行
            - 线程水平的并行
        - 其中，有摩尔定律支持芯片行业的发展：「芯片上的集成晶体管数量每 18 个月增加一倍」。
        - 2000年以前
            - 单处理器性能处于增加状态

        - 同样的串行程序在新的硬件上执行速度变快
        - MHZ、GHZ
    - 发展趋势的变化
        - CPU主频增长越来越缓慢，摩尔定律失效
        - 发展趋势不再是高速的 CPU 主频，而是「多核」。
    - 如何提高CPU的处理速度
        - 1990年之前的解决方式
            - 增加时钟频率
                - 深化流水线：采用更多、更短的流水阶段
                - 芯片的工作温度会过高
            - 推测超标量——多条指令同时执行（指令级并行）
                - 硬件自动找出串行程序中的能够同时执行的独立指令集和
                - 硬件预测分支指令：在分支指令实际发生之前先推测执行
                - 最终出现收益下降
                - 优点：程序员不需要知道这些过程的细节
        - 2000年之后
            - 时钟频率很难再增加
            - 推测超标量触到天花板，收益下降
            - 利用额外的晶体管在芯片上构建更多更简单的处理器
        - 后来发展，延申出了并行计算机和并行计算集群。
        - 未来属于异构、多核的SOC架构，SOC：system on chip
        - 所有超级计算机都在使用众核处理器
- 并行计算机
    - 从硬件角度来看，今天的单个计算机都是并行计算机
        - 多个功能单元
        - 多个执行单元或核心
        - 多个硬件线程
- 并行计算集群：多个单独的计算机通过网络连接起来形成计算集群
    - 每个节点都是一个多处理器并行机
    - 每个计算节点通过Infiniband、网络连接
- **并行计算发展驱动力**
    - Moore's law新解
    - 每两年芯片上的核心数目就会翻倍
    - 时钟频率不再增加甚至更低
    - 需要处理具有很多并发线程的系统
    - 需要处理芯片内并行和芯片之间的并行

    - 需要处理异构和各种规范
- 并行计算不仅仅是硬件



➢ VAX 指令的谬论：

1. 对于高阶抽象只用一条指令表示；
2. 单条硬件指令真的快吗？

➢ RISC指令的设计思想：

1. 全系统设计；
2. 从全局角度设计硬件机制；
3. 跨组件优化

➢ 当前程序员看不到汇编语言，甚至看不到底层C语言。

并行编程！

- 并行编译器的局限
    - 编译器做到了有限的并行检测和程序转换
        - 执行级的并行，只能检测到basic block
        - 少量的嵌入的for
        - 分析技术如：数据依赖分析、指针分析、流敏感分析，难以应用到大规模程序中
    - 比较成功的做法是让人学习如何编写并行程序
- 并行编程的例子
    - 使用 OS 线程编程，可能会变得复杂且容易出错
- **并行编程的难点**
    - 找到尽可能多的并行点
    - **粒度**：函数级并行、线程级并行还是进程级并行
    - 局部性：并行化之后是否能够利用局部数据
    - 负载均衡：不同线程、不同core之间的负载分布
    - 协作与同步
    - 性能模型：所有这些难点让并行编程比串行编程复杂得多
- 并行编程与串行编程的比较
    - 编程代价不同、优势不同
    - 并行编程需要不同的，不熟悉的算法
    - 并行编程必须利用不同的问题抽象
    - 并行程序的行为更为复杂
    - 更难以控制程序不同组件之间的交互

    - 需要掌握更多编程工具、更多知识
- 编写并行程序
    - 任务并行：将一个问题划分成多个不同的子任务分别在不同的core上运行
    - 数据并行：将问题所涉及的数据拆分到不同的core上执行；每个core运行的是相同的程序
- 并行程序之间的协作
    - cores之间通常需要相互协作
    - 通信：一个或多个core会将他们当前计算结果发送到其他core上
    - 负载均衡：在core之间平衡分配任务，这样不至于出现某个core过载的现象，减少尾长
    - 同步：每个core独立运行，在某些点上，确保core之间的状态同步
- Amdahl's law
    - 用于度量并行程序的加速效果，理想的并行加速效果

$$\text{Speedup} = \frac{\text{1-thread execution time}}{\text{n-thread execution time}}$$

    - $speedup = \frac{1}{(1-p)+p/n}$
        - p为并行计算部分所占的比例，n为并行处理结点的个数
    - 理想现实差距太大的原因
        - 应用程序只有一部分可以并行执行
        - 大量的串行代码降低并行性能
- 编程模型

Standardized parallel programming platforms

- **OpenMP**
- **Message Passing Interface (MPI)**
- **Posix Threads (pthreads)**
- **Open Computing Language (OpenCL)**

以上内容整理于 幕布文档