并行算法的本质在于：

利用比较充足的硬件资源，
通过增加总工作量的方法，
减少总的时间消耗。

所以：

- 需要大型的计算硬件支持；
- 并行算法（一般）不能节省计算量；
- 能够节省计算时间；

并行算法的评估

串行算法的评估仅仅依赖于

运行时间 / 问题规模

并行算法的运行时间则依赖于

问题规模
进程个数
进程的相对速度
进程间通信速度
… …

但是，对于并行算法，还是能够比较客观的进行评估。

加速比：

加速比＝串行运行时间／运行时间 x 进程个数

加速比作为进程个数的函数不是常数；
加速比也是依赖于机器硬件；

$$\text{加速比} \leq 1$$

造成加速比小于 1 的原因：

- 负载不平衡；
- 进程交互；
- 同步；
- 算法缺陷：
  - 最好的串行算法几乎总是不能被顺利地并行化，从而能够并行的总是一些效率不是最高的算法。

加速比在实际情况中能观察到所谓超线性情况，即

$$\text{加速比} > 1$$

的情况，这样的情形发生的原因一般来说是：

- 由于机器的硬件架构造成的，比如说Cache；
- 算法花费的时间具有某种不确定性；

例子：

树结构中的特定元素搜索问题；

加速比

作为问题规模的函数，一般是增函数；

作为进程个数的函数，一般是减函数；

所以对于给定的问题规模而言，

计算时间存在一个最小值

t_s：一个通信的启动时间

t_w：单字通信时间

p：进程数；

m：消息字数；

超立方体拓扑结构

All-to-One 归约和 One-to-All 广播的时间花费

$$T=(t_s+t_w m)\log p$$

All-to-All 花费时间：

$$T=\sum_{i=1}^{\log p}\left(t_s+2^{i-1}t_w m\right)=t_s\log p+t_w m(p-1)$$

稠密矩阵的相关算法

矩阵 x 向量

矩阵按行分配给进程；

n 行 n 进程；

少于 n 进程：每进程 n/p 行；

$$T=\frac{n^2}{p}+t_s\log p+t_w n$$

矩阵按二维块分配给进程：$\sqrt{p}\times\sqrt{p}$

nxn 进程；

少于 nxn 进程：

$$T=\frac{n^2}{p}+t_s\log p+t_w\frac{n\log p}{\sqrt{p}}$$

对于非常大的 n 和 p，二维分配方式略好一点。

矩阵 x 矩阵

标准串行乘法 C= AxB

```
for i=1,n
    for j=1,n
        C_ij = 0
        for k=1,n
            C_ij = C_ij + A_ik B_kj
        end for
    end for
end for
```

分块矩阵乘法的算法

先将矩阵分成(n/p)x(n/p)的分块矩阵，然后

for i=1,p
　for j=1,p
　　$C_{ij}=(0)$
　　for k=1,q
　　　$C_{ij}=C_{ij}+A_{ik}B_{kj}$
　　end for
　end for
end for

初始状态：

　　每个节点拥有A和B的(i,j)子块

要求计算：

　　结果的(i,j)子块

方案：

1. A按行广播， B按列广播；
2. 在每个节点上做乘法；

$$T=\frac{n^3}{p}+t_s\log p+2t_w\frac{n^2}{\sqrt{p}}$$

Cannon 算法

节省内存，使用移位操作

$$T=\frac{n^3}{p}+2t_s\sqrt{p}+2t_w\frac{n^2}{\sqrt{p}}$$

DNS(Dekel, Nassimi and Sahni) 算法

可以使用最多 nxnxn 个进程，并行化程度更高；
如果少于 nxnxn 个进程，可以用于实际计算；

$$T=\frac{n^3}{p}+t_s\log p+t_w\frac{n^2\log p}{p^{2/3}}$$

求解稠密线性方程组

$$Ax=b$$

LU 分解

> 简单按行分解的并行化；

> 流水线形式：

1．如果有数据需要传送，则传送数据；
2．如果有计算需要做，则做计算；
3．等待收到数据，以便转到１或２；

> 少于 n 个进程；

按二维方式分块：

- 简单并行处理；
- 流水线形式；
- 少于 nxn 个进程；

卷帘式存储方案：

6.2 节；

选主元：

- 行选主元、列选主元
- 一维存储方式、二维分块存储方式
- 显式交换、隐式交换

解上三角阵：

数值精度：