并行计算

十三、并行程序设计基础

并行程序设计基础

13.1 并行程序设计概述

13.2 并行程序设计模型

并行程序设计难的原因

- 技术先行, 缺乏理论指导
  - 程序的语法/语义复杂, 需要用户自己处理
  - 任务/数据的划分/分配
  - 数据交换
  - 同步和互斥
- 并行语言缺乏代可扩展和异构可扩展, 程序移植困难, 重写代码难度太大
- 环境和工具缺乏较长的生长期, 缺乏代可扩展和异构可扩展

并行语言的构造方法

串行代码段

```
for ( i= 0; i<N; i++ ) A[i]=b[i]*b[i+1];
for (i= 0; i<N; i++) c[i]=A[i]+A[i+1];
```

(a) 使用库例程构造并行程序

```
id=my_process_id();
p=number_of_processes();
for ( i= id; i<N; i=i+p) A[i]=b[i]*b[i+1];
barrier();
for (i= id; i<N; i=i+p) c[i]=A[i]+A[i+1];
```

例子：**MPI,PVM, Pthreads**

(b) 扩展串行语言

my_process_id,number_of_processes(), and barrier()

```
A(0:N-1)=b(0:N-1)*b(1:N)
c=A(0:N-1)+A(1:N)
```

例子: **Fortran 90**

(c) 加编译注释构造并行程序的方法

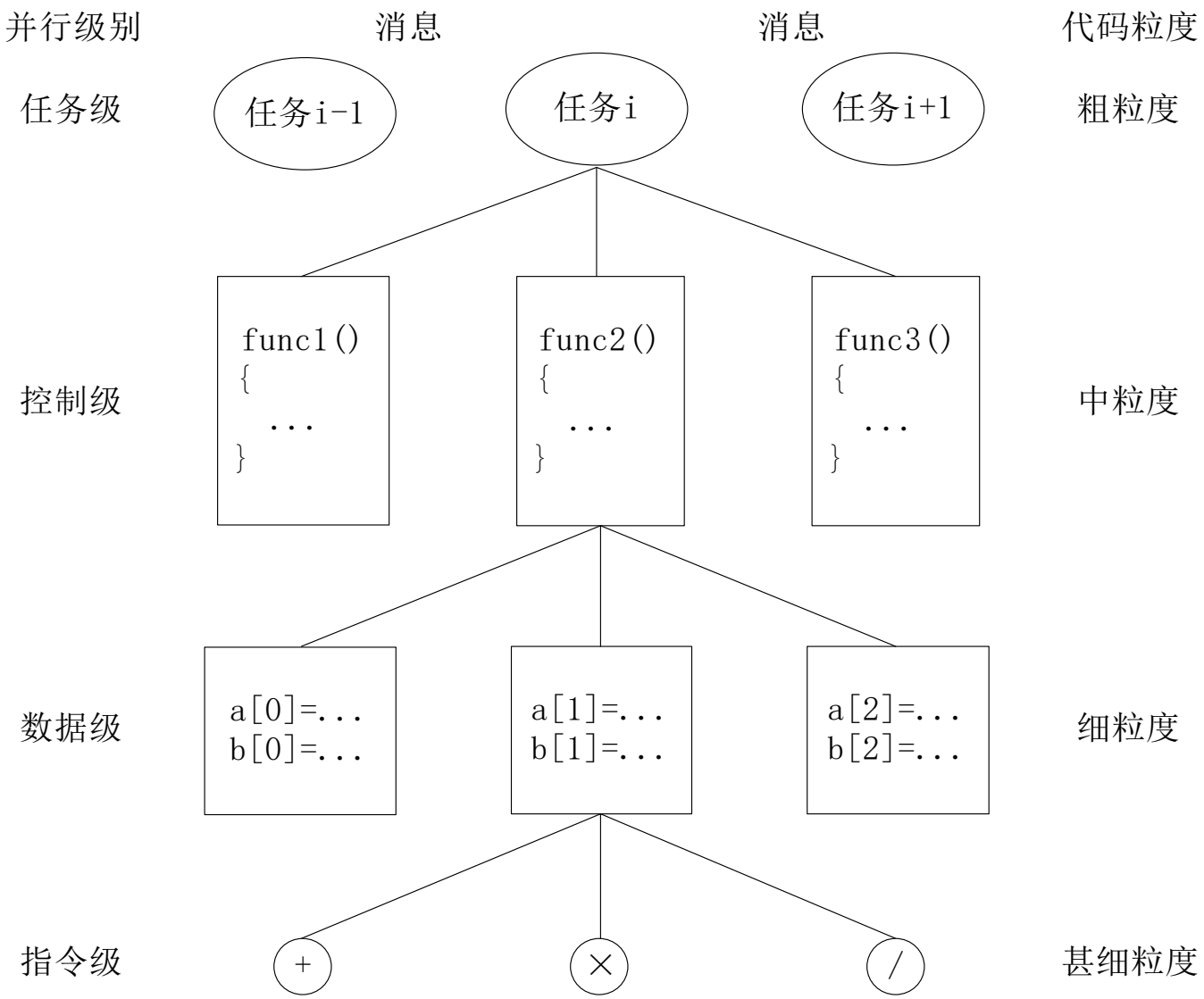
```
#pragma parallel
#pragma shared(A,b,c)
#pragma local(i)
{
# pragma pfor iterate(i=0;N;1)
for (i=0;i<N;i++) A[i]=b[i]*b[i+1];
# pragma synchronize
# pragma pfor iterate (i=0; N; 1)
for (i=0;i<N;i++)c[i]=A[i]+A[i+1];
}
```

例子：**SGI power C**

并行语言的构造方法

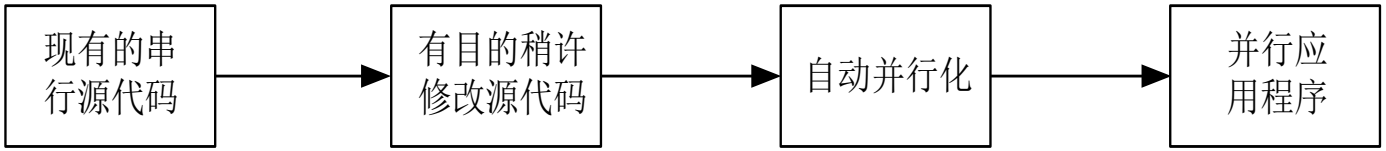
方法	实例	优点	缺点
库例程	MPI, PVM	易于实现，不需要新编译器	无编译器检查，分析和优化
扩展	Fortran90	允许编译器检查、分析和优化	实现困难,需要新编译器
编译器注释	SGI powerC, HPF	介于库例程和扩展方法之间，在串行平台上不起作用.	

并行层次与代码粒度

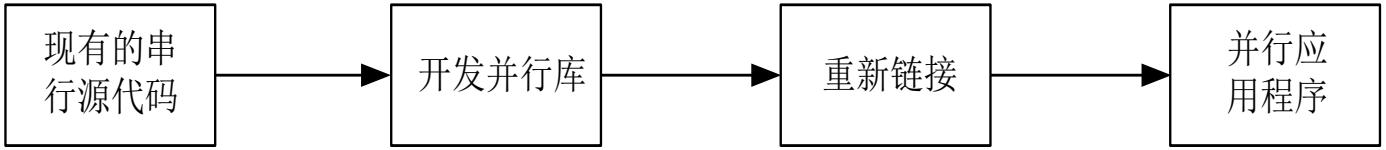

并行级别
消息
消息
代码粒度
任务级
任务i-1
任务i
任务i+1
粗粒度
控制级
func1()
{
...
}
func2()
{
...
}
func3()
{
...
}
中粒度
数据级
a[0]=...
b[0]=...
a[1]=...
b[1]=...
a[2]=...
b[2]=...
细粒度
指令级
+
×
/
甚细粒度

并行层次	粒度（指令数）	并行实施	编程支持
甚细粒度指令级并行	几十条，如多指令发射、内存交叉存取	硬件处理器	
细粒度数据级并行	几百条，如循环指令块	编译器	共享变量
中粒度控制级并行	几千条，如过程、函数	程序员（编译器）	共享变量、消息传递
粗粒度任务级并行	数万条，如独立的作业任务	操作系统	消息传递

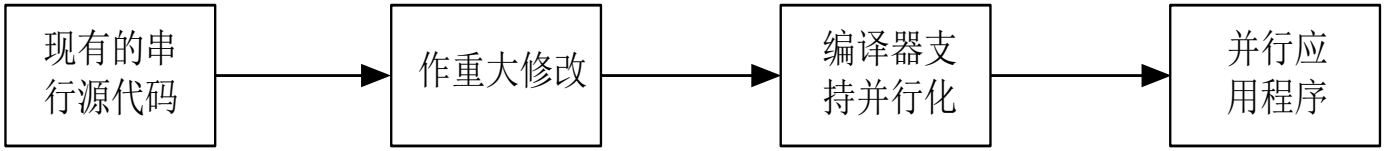
并行程序开发策略



(a) 自动并行化



(b) 并行库



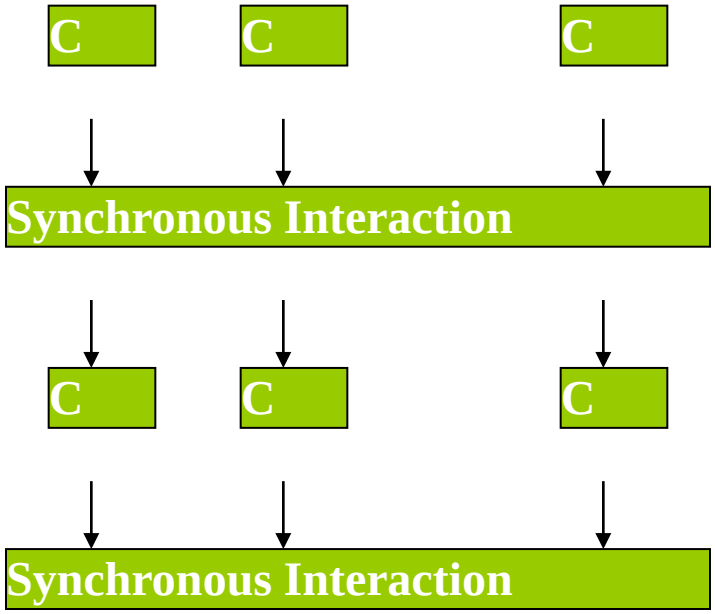
(c) 重新编写并行代码

并行编程风范

- 相并行（Phase Parallel）
- 分治并行（Divide and Conquer Parallel）
- 流水线并行（Pipeline Parallel）
- 主从并行（Master-Slave Parallel）
- 工作池并行（Work Pool Parallel）

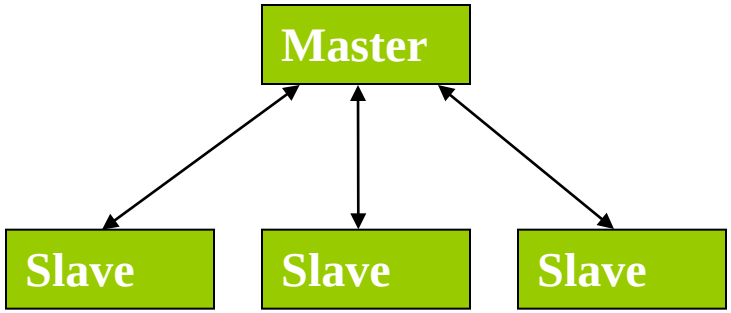
相并行（Phase Parallel）

- 一组超级步（相）
- 步内各自计算
- 步间通信、同步
- BSP（4.2.3）
- 方便查错和性能分析
- 计算和通信不能重叠



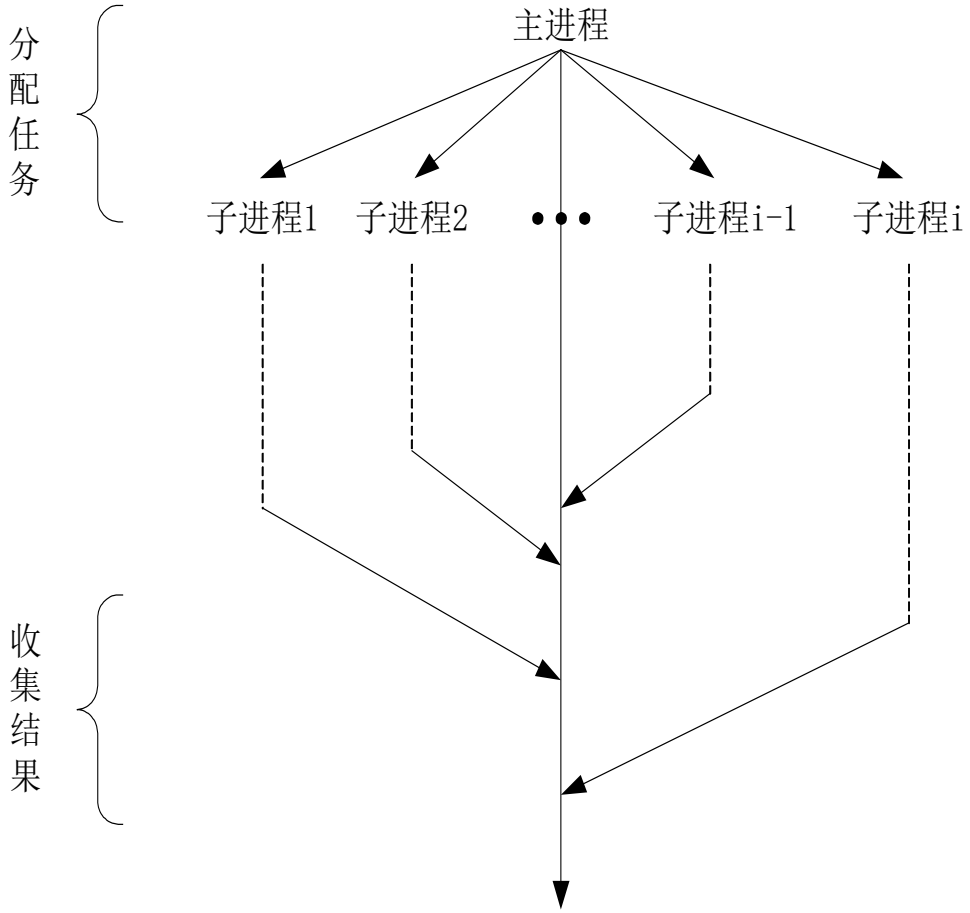
主－从并行（Master-Slave Parallel）

- 主进程：串行、协调任务
- 子进程：计算子任务
- 划分设计技术（ 6.1）
- 与相并行结合
- 主进程易成为瓶颈



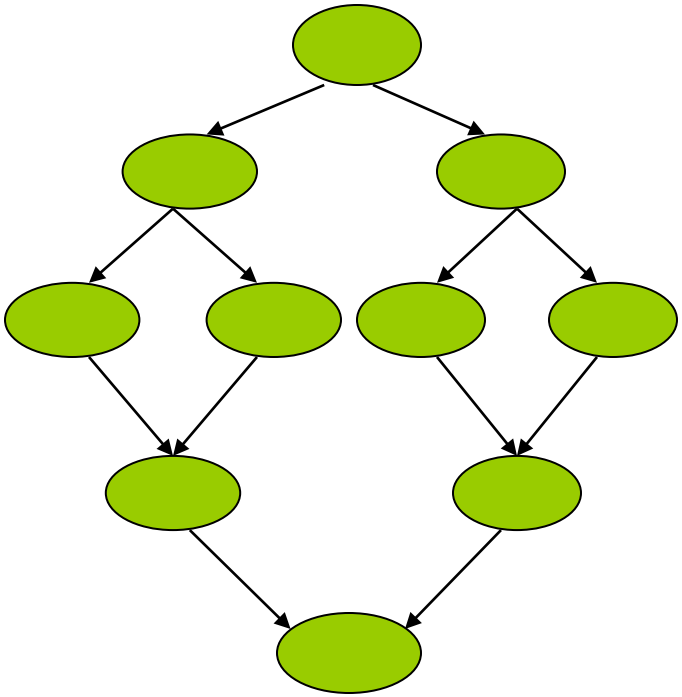
主-从式（Master-Slave）

- 基本思想是将一个待求解的任务分成一个主任务（主进程）和一些从任务（子进程）。主进程负责将任务的分解、派发和收集诸子任务的求解结果并最后汇总得到问题的最终解。诸子进程接收主进程发来的消息；并行进行各自计算；向主进程发回各自的计算结果。



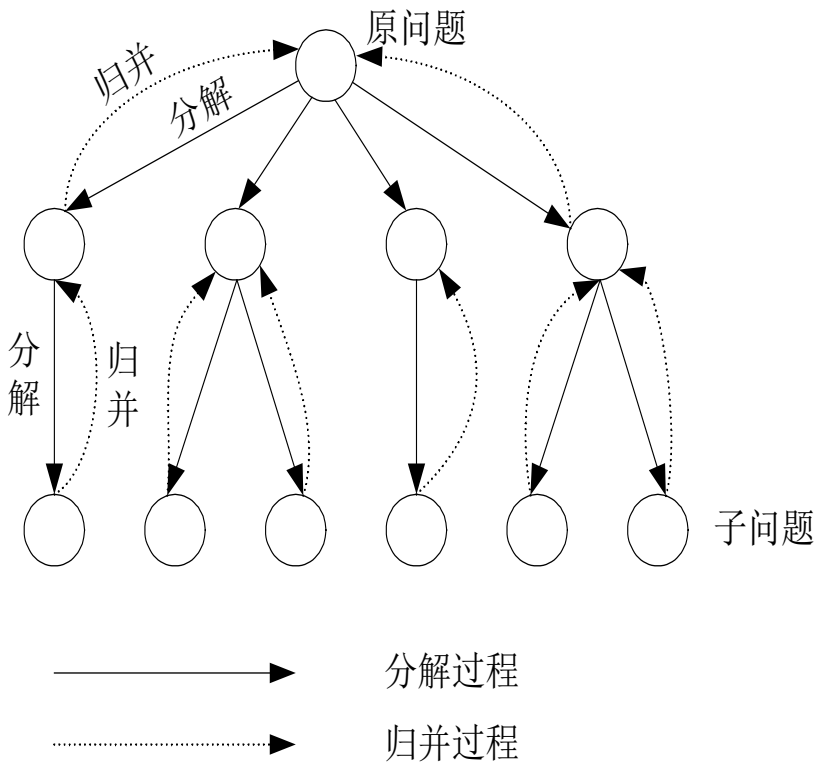
分治并行（Divide and Conquer Parallel）

- 父进程把负载分割并指派给子进程
- 递归
- 重点在于归并
- 分治设计技术（6.2）
- 难以负载平衡



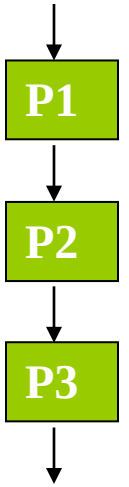
分治策略（Divide and Conquer）

- 其基本思想是将一个大而复杂的问题分解成若干个特性相同的子问题分而治之。若所得的子问题规模仍嫌过大，则可反复使用分治策略，直至很容易求解诸子问题为止。问题求解可分为三步：①将输入分解成若干个*规模近于相等*的子问题；②同时递归地求解诸子问题；③归并各子问题的解成为原问题的解。



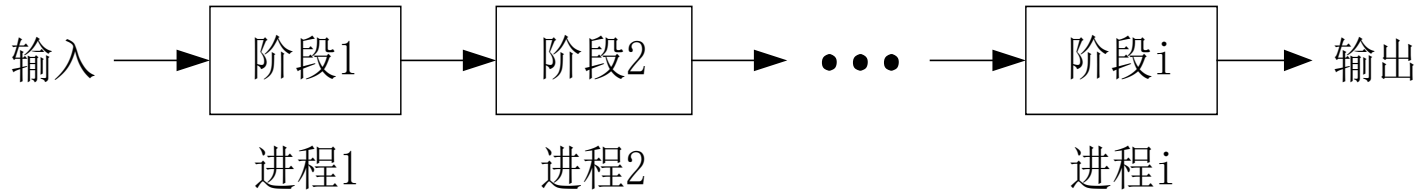
流水线并行（Pipeline Parallel）

- 一组进程
- 流水线作业
- 流水线设计技术（6.5）



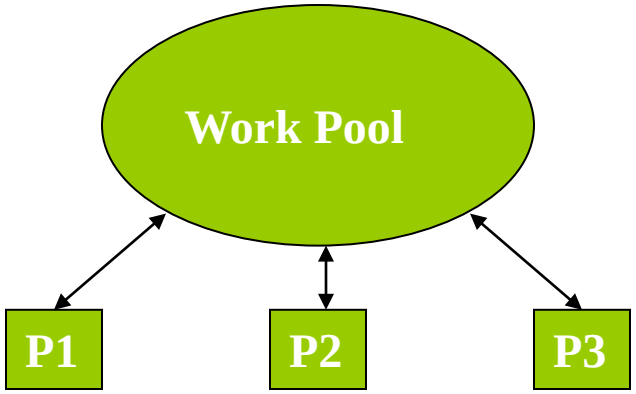
数据流水线（Data Pipelining）

- 其基本思想是将各计算进程组织成一条流水线，每个进程执行一个特定的计算任务，相应于流水线的一个阶段。一个计算任务在功能上划分成一些子任务（进程），这些子任务完成某种特定功能的计算工作，而且一旦前一个子任务完成，后继的子任务就可立即开始。在整个计算过程中各进程之间的通信模式非常简单，仅发生在相邻的阶段之间，且通信可以完全异步地进行。



工作池并行（Work Pool Parallel）

- 初始状态：一件工作
- 进程从池中取任务执行
- 可产生新任务放回池中
- 直至任务池为空
- 易于负载平衡
- 临界区问题（尤其消息传递）



并行程序设计基础

12.1 并行程序设计概述

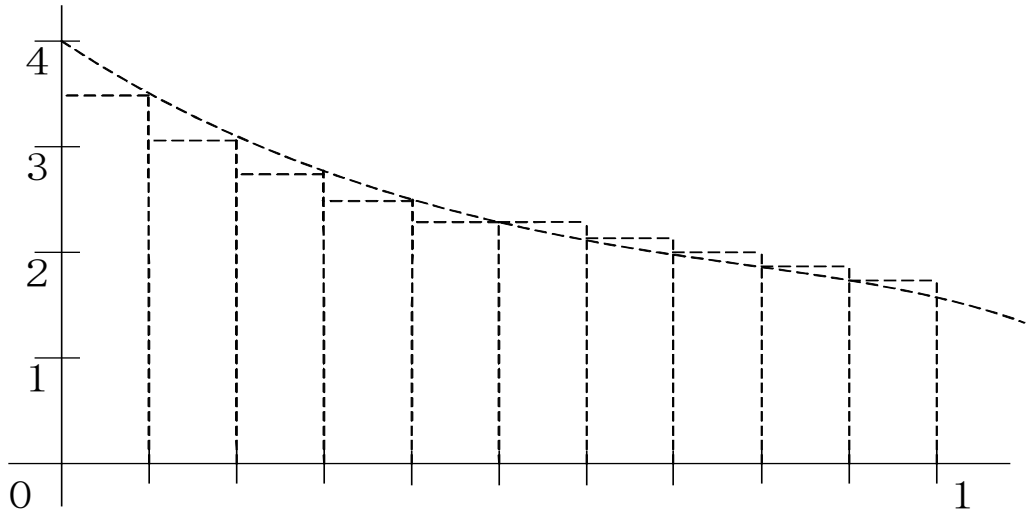
12.2 并行程序设计模型

并行程序设计模型

- 隐式并行（Implicit Parallel）
- 数据并行模型（Data Parallel）
- 消息传递模型（Message Passing）
- 共享变量模型（Shared Variable）

π的计算

$$\pi = \int_0^1 \frac{4}{1+x^2}\,dx \approx \sum_{0 \le i \le N} \frac{4}{1+\left(\frac{i+0.5}{N}\right)^2} \cdot \frac{1}{N}$$



计算π的串行C代码

```
#define  N  1000000
main()   {
    double local, pi = 0.0, w;
    long  i;
    w=1.0/N;
    for (i = 0; i<N; i ++) {
          local = (i + 0.5)*w;
          pi = pi + 4.0/(1.0+local * local);
          }
    printf("pi is %f \n", pi *w);
}
```

隐式并行（Implicit Parallel）

- 概况：
  - 程序员用熟悉的串行语言编程
  - 编译器或运行支持系统自动转化为并行代码
- 特点：
  - 语义简单
  - 可移植性好
  - 单线程，易于调试和验证正确性
  - 效率很低

数据并行（Data Parallel）

- 概况：
  - SIMD的自然模型，也可运行于SPMD、MIMD机器上
  - 局部计算和数据选路操作
  - 适合于使用规则网络，模板和多维信号及图像数据集来求解细粒度的应用问题
  - 数据并行操作的同步是在编译时而不是在运行时完成的
- 特点：
  - 单线程
  - 并行操作于聚合数据结构（数组）
  - 松散同步
  - 全局命名空间
  - 隐式相互作用
  - 隐式/半隐式数据分布

计算π的数据并行代码

```
main( )

{
        long i，j，t，N=100000；
            double local [N]，temp [N]，pi，w；
    A：  w=1.0/N；
    B：  forall (i=0；i<N ；  i++){
       P：  local[i]=(i+0.5)*w；
      Q：  temp[i]=4.0/(1.0+local[i]*local[i])；
       }
    C：     pi = sum (temp)；
    D：    printf (“pi is %f \ n”，pi * w )；
  }  / *main( ) * /
```

消息传递（Message Passing）

- 概况：
  - MPP, COW的自然模型，也可应用于共享变量多机系统，适合开发大粒度的并行性
  - 广泛使用的标准消息传递库MPI和PVM
- 特点：
  - 多线程
  - 异步并行性
  - 分开的地址空间
  - 显式相互作用
  - 显式数据映射和负载分配
  - 常采用SPMD形式编码

计算π的MPI代码

```
# define  N  100000
  main ( ){
     double  local=0.0，pi，w，temp=0.0；
        long i ，taskid，numtask；
  A：   w=1.0/N；
        MPI_ Init(&argc，& argv)；
        MPI _Comm _rank (MPI_COMM_WORLD，&taskid)；
        MPI _Comm _Size (MPI_COMM_WORLD，&numtask)；
  B：  for (i= taskid；  i< N；  i=i + numtask){
        P：  temp = (i+0.5)*w；
        Q：  local=4.0/(1.0+temp*temp)+local；
        }
  C：   MPI_Reduce (&local,&pi,1,MPI_Double,MPI_SUM,0,
                  MPI_COMM_WORLD)；
  D：  if (taskid = =0) printf(“pi is %f \ n”，pi* w)；
        MPI_Finalize ( ) ；
        } / * main ( )*/
```

共享变量（Shared Variable）

- 概况：
  - PVP, SMP, DSM的自然模型
- 特点：
  - 多线程：SPMD, MPMD
  - 异步
  - 单一共享地址空间
  - 显式同步
  - 隐式/隐式数据分布
  - 隐式通信（共享变量的读/写）

计算π的共享变量程序代码

```
# define  N  100000
  main ( ){
        double  local，pi=0.0 ，w；
        long  i ；
  A ：    w=1. 0/N；
  B ：    # Pragma Parallel
        # Pragma Shared (pi，w)
        # Pragma Local (i，local)
         {
         # Pragma pfor iterate(i=0；N；1)
         for (i=0；i<N，i++){
         P：  local = (i+0.5)*w；
         Q：  local=4.0/(1.0+local*local)；
         }
  C ：    # Pragma Critical
        pi =pi +local ；
        }
  D：     printf (“pi is %f \ n”，pi *w)；
      }/ *main( ) */
```

三种显式并行程序设计模型主要特性

特 性	数据并行	消息传递	共享变量
控制流（线）	单线程	多线程	多线程
进程间操作	松散同步	异步	异步
地址空间	单一地址	多地址空间	单地址空间
相互作用	隐式	显式	显式
数据分配	隐式或半隐式	显式	隐式或半隐式

并行编程语言和环境概述

早期机制		近代机制		并行说明性机制	
共享存储	分布存储	共享存储	分布存储	逻辑语言	函数语言
Semaphores （信号灯） CCRs （条件 临界区） Monitors （管理）	Sockets （套接字） RPCs （远程过程 调用） Rendezvous （汇合）	Pthreads Java Threads SHMEM OpenMP HPF （虚拟 共享） Pthreads	Ada MPI PVM DCE dist. Java	IC-Prolog PARLOG GHCs	Multilisp Sisal