并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

概要

- 第三篇 并行编程
  - **第十三章 并行程序设计基础**
  - 第十四章 共享存储系统并行编程
  - 第十五章 分布存储系统并行编程
  - 补充章节1 GPU并行编程
  - 补充章节2 关于并行编程的更多话题

第十三章 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

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

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（函数调用）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, Pthread

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];
```

(b) 扩展串行语言

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

例子: Fortran 90, matlab

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];
```

(c) 加编译注释构造并行程序的方法

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

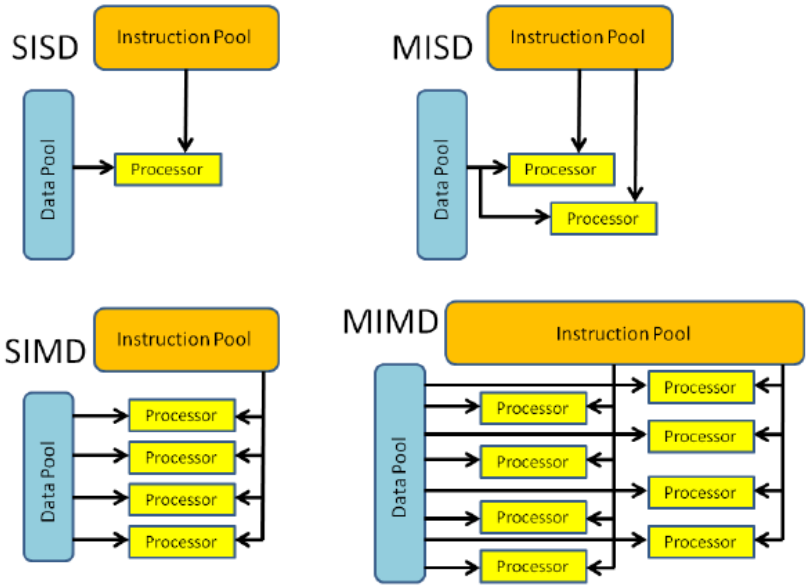
例子：OpenMP

并行语言的构造方法

方法	实例	优点	缺点
库例程	MPI	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	OpenMP	介于库例程和扩展方法之间，在串行平台上不起作用.	

Flynn分类法的扩展

- Flynn分类法是针对硬件结构而言的
- 计算模型是虚拟硬件结构，可以沿用Flynn分类法
  - RAM, PRAM, APRAM, BSP, LogP ……
- 从并行编程的角度，需要做适当扩展



Flynn分类法的扩展

- SIMD: 所有进程在同一时间执行相同的指令
  - 软件上很难实现（相当于每条指令后都有同步）
- MIMD:各个进程在同一时间可执行不同的指令
  - SPMD （Single Program Multiple Data）

  各个进程是同构的，多个进程对不同的数据执行同一份代码 (的**相同或不同的指令**)
  - MPMD（Multiple Program Multiple Data）

  各个进程是异构的， 多个进程执行不同的代码
- 概念梳理
  - SIMD：硬件支持、细粒度
  - SPMD：软件实现，粗粒度
  - 虽然SPMD和SIMD比较像，但SPMD属于MIMD的一种

SPMD程序的构造方法

用单代码描述SPMD

要说明以下SPMD程序:

```
parfor (i=0; i<=N, i++)  foo(i);
```

用户需写一个以下程序:

```
pid=my_process_id();
numproc=number_of _processes();
parfor (i=pid; i<=N, i=i+numproc)
foo(i);
```

此程序经编译后生成可执行程序A, 用shell脚本将它加载到N个处理结点上:

```
run A –numnodes N
```

用数据并行描述SPMD

要说明以下SPMD程序:

```
parfor ( i=0; i<=N, i++) {
C[i]=A[i]+B[i];
}
```

用户可用一条数据赋值语句:

```
C=A+B
```

或

```
forall (i=1,N) C[i]=A[i]+B[i]
```

MPMD程序的构造方法

用多代码描述MPMD

对于并行执行的S1, S2, S3三个任务，需分别编写三个可执行程序S1, S2, S3，用Shell脚本将他们加载到三个处理结点上：

Run S1 on node1

Run S2 on node2

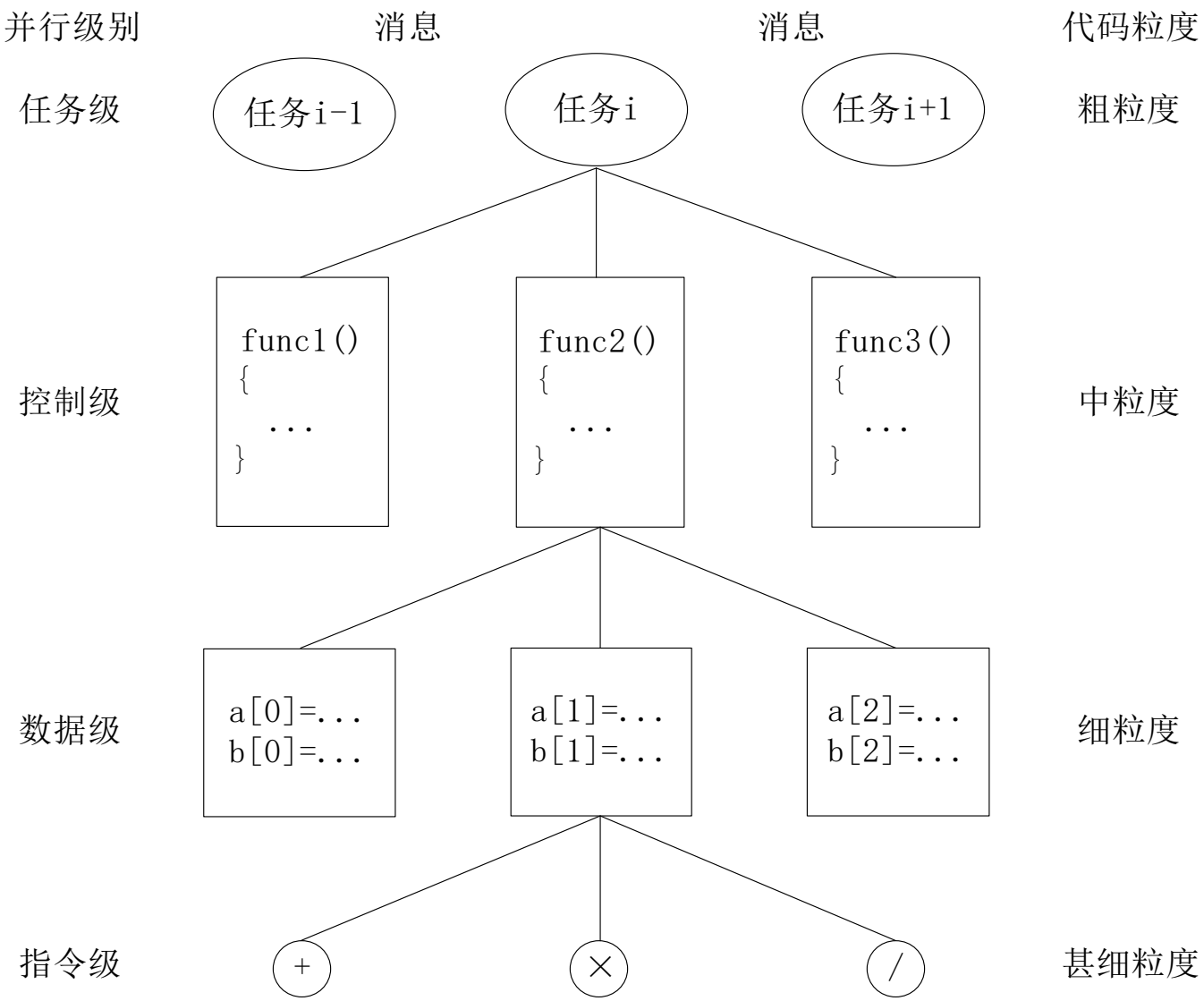
Run S3 on node3

用SPMD描述MPMD

对于并行执行的S1, S2, S3三个任务，可用SPMD程序构造：

```
parfor (i=0; i<3, i++) {
    if (i==0) S1
    if (i==1) S2
    if (i==2) S3
}
```

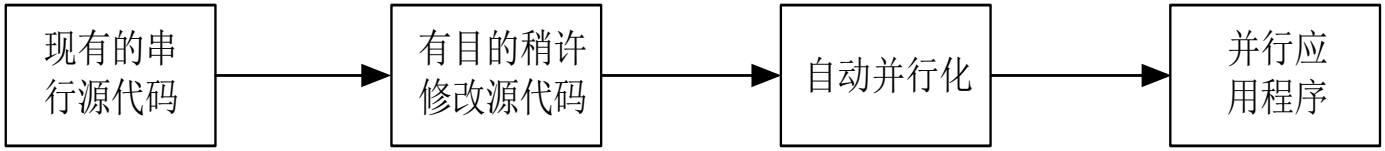
并行层次与代码粒度


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

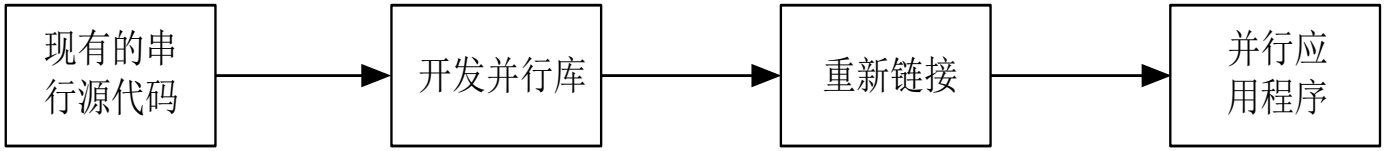
并行层次与代码粒度

并行层次	粒度（指令数）	并行执行	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

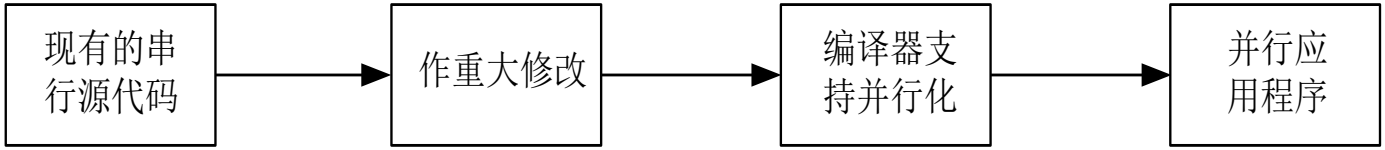
并行程序开发策略



(a) 自动并行化



(b) 并行库



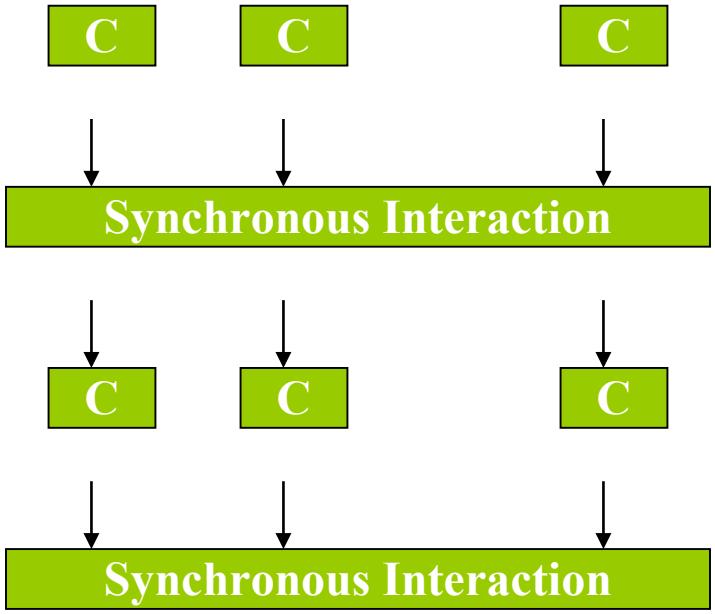
(c) 重新编写并行代码

并行编程范式
Parallel Programming Paradigms

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

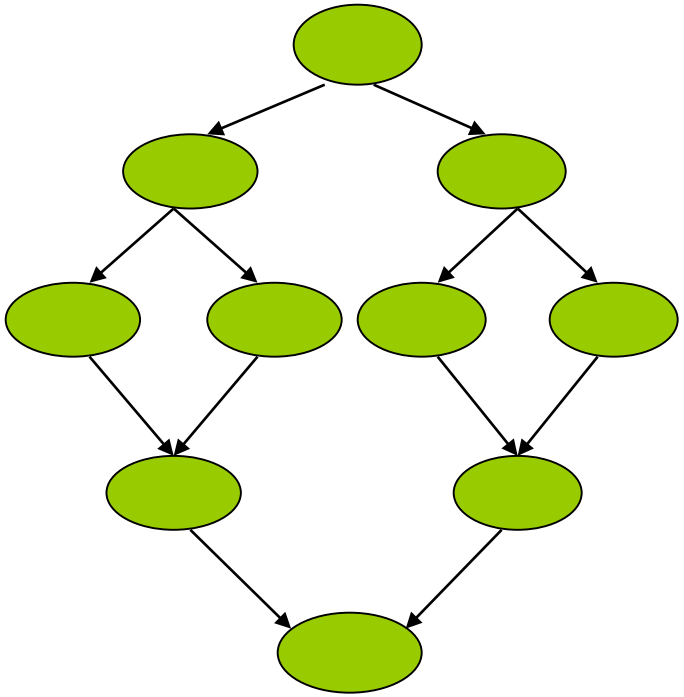
相并行

- 一组超级步（相）
- 步内各自计算
- 步间通信、同步
- BSP（教材5.2.3）
- 方便查错和性能分析
- 计算和通信不能重叠



分治并行

- 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（教材7.2）
- 难以负载平衡



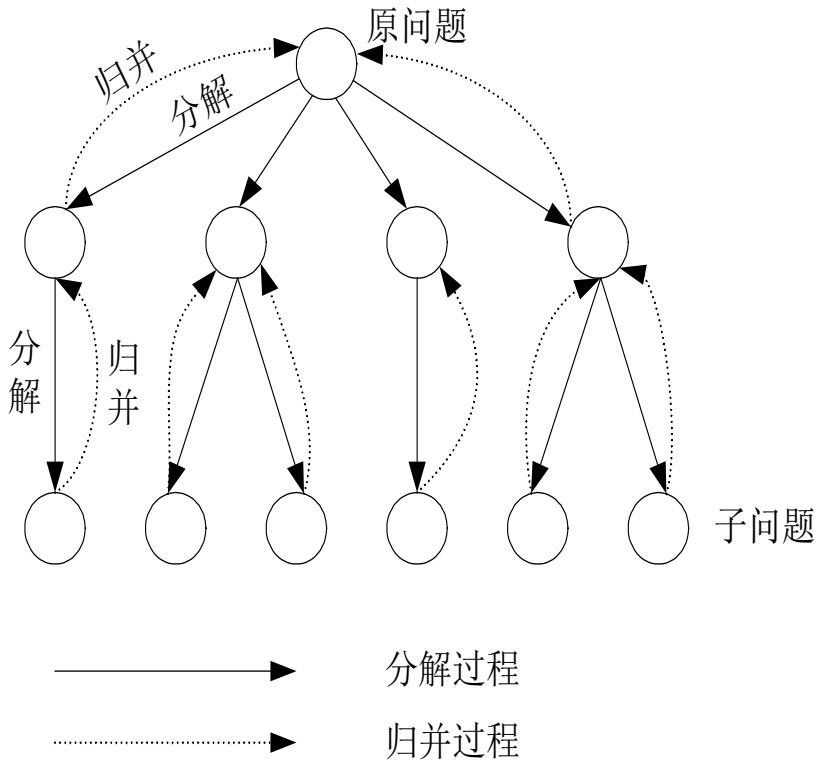
分治并行

- 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。

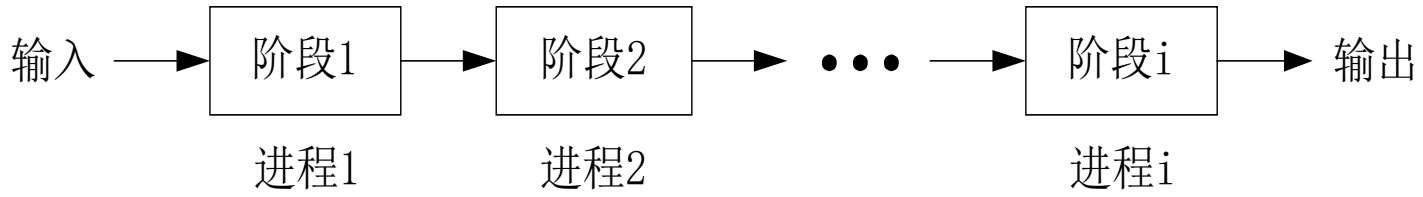


流水线并行

- 一组进程
- 流水线作业
- 流水线设计技术（教材7.5）

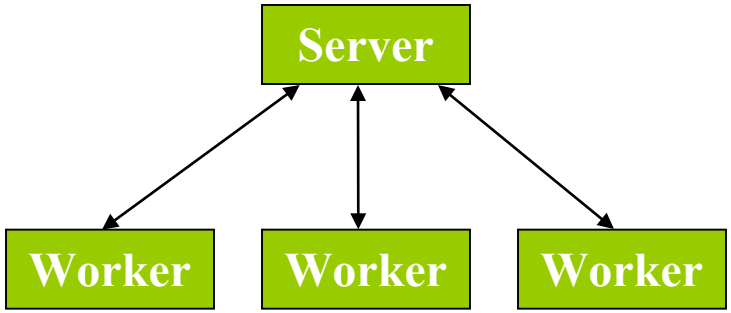
流水线并行

- 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仅发生在相邻的阶段之间，且通信可以异步地进行。



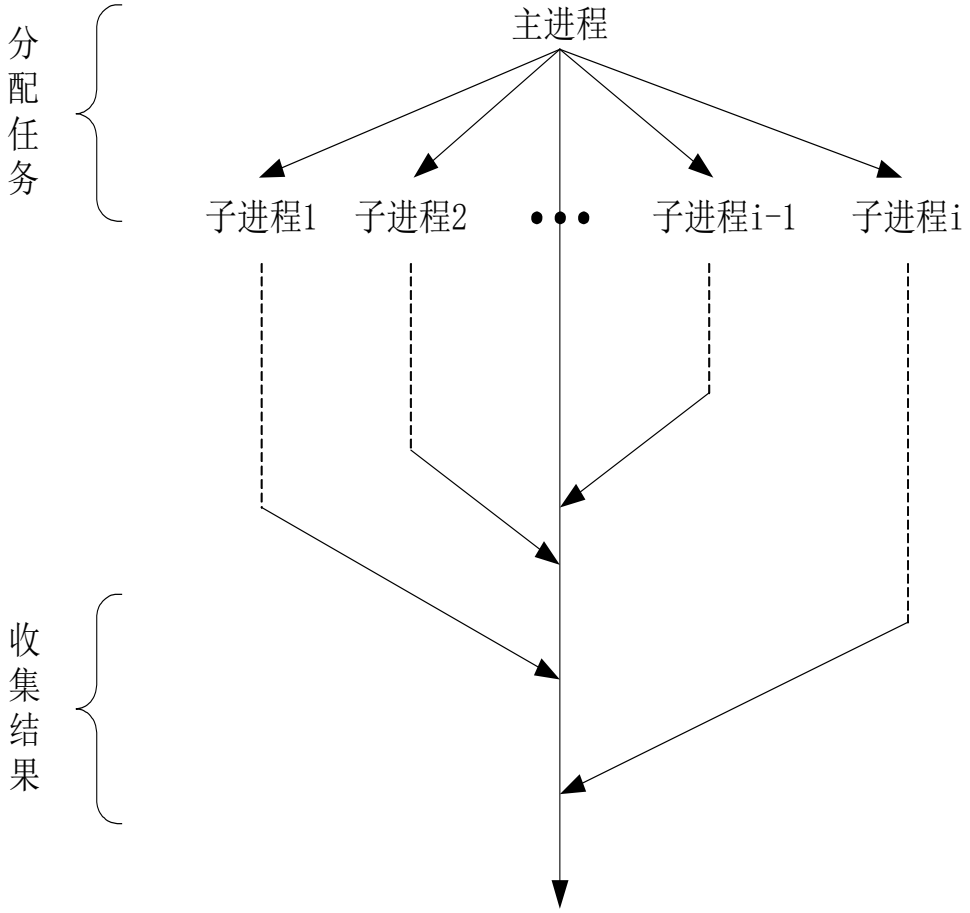
主从并行

- 主进程：串行、协调任务
- 子进程：计算子任务
- 划分设计技术（教材7.1）
- 与相并行结合
- 主进程易成为瓶颈



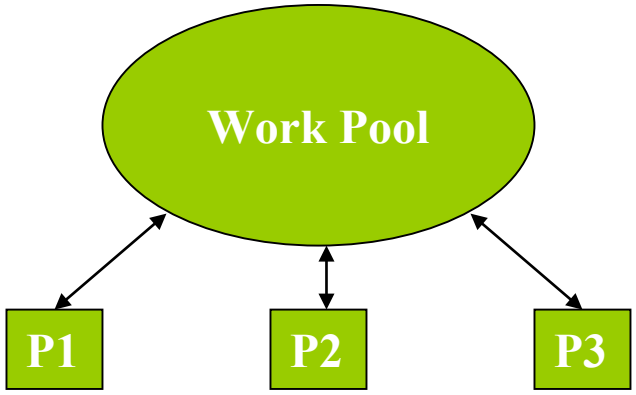
主从并行

- 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。
- 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。



工作池并行

- 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）



第十三章 并行程序设计基础

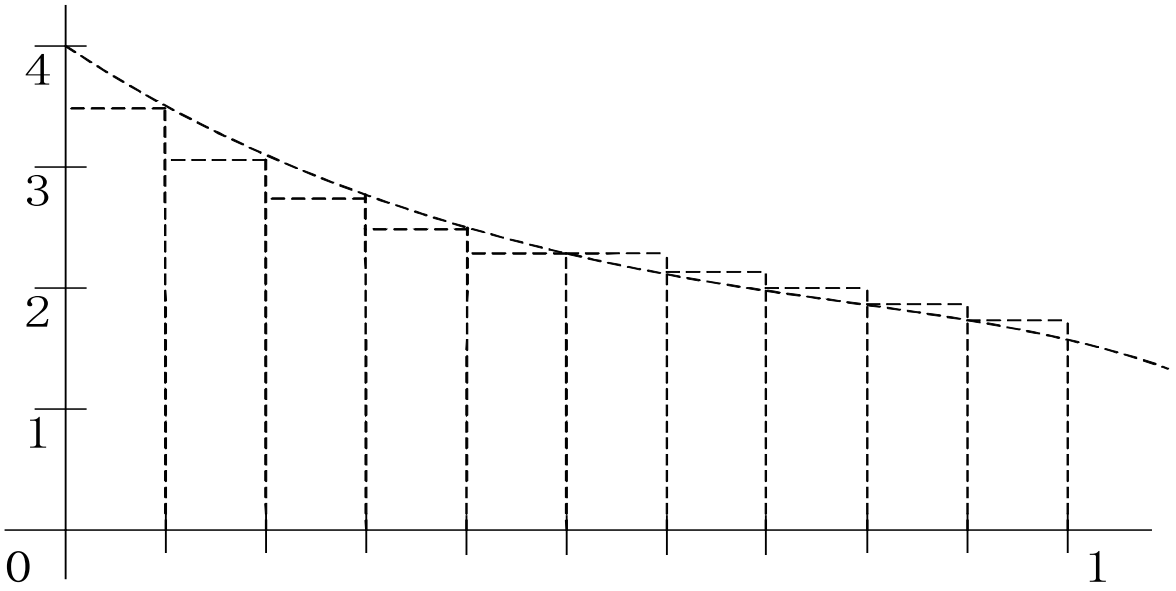
- 13.1 并行程序设计概述
- **13.2 并行程序设计模型**

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

- 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
  - 易于调试和验证正确性
  - 效率通常可进一步优化

隐式并行

python 复制

```python
import numpy as np

# 定义梯形的数量（梯形越多，近似越好）
n = 1000000  # 这是一个很大的数字，用于细粒度近似

# 从0到1生成n个等间距的值
x = np.linspace(0, 1, n+1)

# 计算每个梯形的高（函数在每点的值）
y = 1 / (1 + x**2)

# 计算每个梯形的面积并将它们相加
# （区域的两端被减半因为它们只属于一个梯形）
area = (y[0] / 2 + y[-1] / 2 + y[1:-1].sum()) * (1 / n)

# 乘以4以得到π，因为从0到1对arctan(x)的积分是π/4
pi_approx = 4 * area

# 显示结果
print(f"近似的π值是: {pi_approx}")
```

每一句都是并行计算

数据并行

需要注意的是，教材中的数据并行是一种狭义的定义，对应早年的FORTRAN 90等编程语言。近年来在分布式机器学习等领域中的“数据并行”并不要求编译时同步、单线程、隐式同步、全局命名空间、隐式分配等特点。

- SIMD的对应模型
- 适合在规则网络，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（相对地，SIMD每条指令都同步，是严格同步）
  - 隐式同步
  - 全局变量命名空间（即单一共享地址空间）
  - 隐式的数据分配

数据并行

python 复制

```python
import numpy as np

# 定义梯形的数量（梯形越多，近似越好）
n = 1000000  # 这是一个很大的数字，用于细粒度近似

# 从0到1生成n个等间距的值
x = np.linspace(0, 1, n+1)

# 计算每个梯形的高（函数在每点的值）
y = 1 / (1 + x**2)

# 计算每个梯形的面积并将它们相加
# （区域的两端被减半因为它们只属于一个梯形）
area = (y[0] / 2 + y[-1] / 2 + y[1:-1].sum()) * (1 / n)

# 乘以4以得到π，因为从0到1对arctan(x)的积分是π/4
pi_approx = 4 * area

# 显示结果
print(f"近似的π值是: {pi_approx}")
```

数据并行
（隐式并行通常是数据并行）

消息传递

- MPP, COW的对应模型，也可应用于共享变量多机系统，适合粗粒度的并行
- 广泛使用的标准消息传递库是MPI
- 特点：
  - 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形式编写代码

消息传递

```
# 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

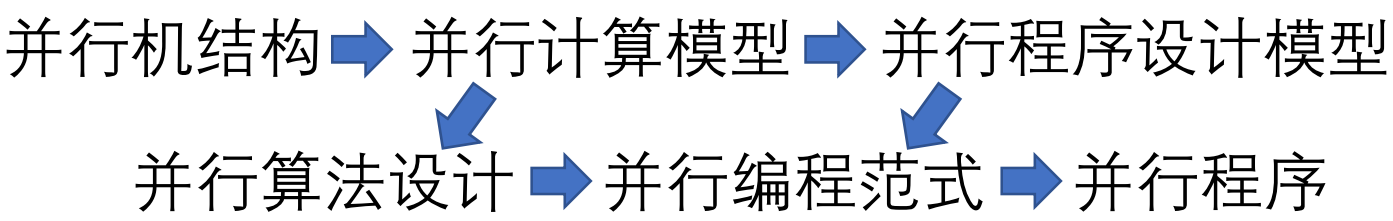
- PVP, SMP, DSM对应的模型

- 特点:
  - 多线程：SPMD, MPMD
  - 异步
  - 单一共享地址空间
  - 显式同步
  - 隐式数据分配
  - 隐式通信（共享变量的读/写）

共享变量

```
# 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( ) */
```

梳理：概念之间的关联性



并行机结构：SMP, PVP, DSM, MPP, COW

并行计算模型 (虚拟并行机)：PRAM, APRAM, BSP, LogP

并行算法设计：分治、平衡树、倍增、流水线

并行程序设计模型：隐式并行、数据并行、共享变量、消息传递

并行编程范式：相并行、分治、流水线、主从、工作池