并 行 计 算

——结构•算法•编程

主讲教师：谢磊

十二、并行程序设计基础

并行程序设计基础

* 12.1 并行程序设计概述
* 12.2 进程
* 12.3 线程
* 12.4 同步
* 12.5 通信
* 12.6 并行程序设计模型

并行程序设计概述

1 并行程序设计难的原因
2 并行语言的构造方法
3 模型和语言的现状和分类
4 模型和语言的评价标准
5 并行性问题
6 交互/通信问题
7 五种并行编程风范
8 计算圆周率的样本程序

1 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
  - 性能平衡
- 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

2 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**

2 并行语言的构造方法

三种并行语言构造方法比较

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.	

3 模型和语言的现状和分类

硬件结构抽象模型(自然模型)

- 共享存储的模型和语言(适于PVP, SMP, DSM)
  - X3H5, Pthread
  - OpenMP
- 消息传递的模型和语言(适于MPP, Cluster, COW)
  - MPI
  - PVM
- 数据并行的模型和语言(适于在MPP/Cluster上实现SPMD应用)
  - Fortran 90
  - HPF(High Performance Fortran)

3　模型和语言的现状和分类

依据对并行性、分解、映射、通信、同步表达的抽象程度进行分类

- 完全抽象的并行模型
- 显式地表达并行，程序分解、映射、通信和同步是隐式的.
- 显式地表达并行和分解，映射、通信和同步是隐式的
- 显式地表达并行、分解和映射，通信和同步是隐式的
- 显式地表达并行、分解、映射和通信，同步是隐式的
- 显式地表达所有的东西

依据抽象程度进行分类

3 模型和语言的现状和分类

完全抽象的并行模型	
动态结构	• *Higher-order Functional-Haskell Concurrent Rewriting:* OBJ, Maude • *Interleaving:* Unity • *Implicit Logic Languages:* PPP, AND/OR, REDUCE/OR, Opera, Palm, concurrent constraint languages
静态结构	• *Algorithmic Skeletons:* P3L, Cole, Darlington
静态和通信受限的结构	• *Homomorphic Skelletons:* Brid-Meertens Formalism • *Celllular Processing Languages:* Cellang, Carpet, CDL, Ceprol • *Crystal*

依据抽象程度进行分类

3 模型和语言的现状和分类

显式地表达并行，但程序分解为线程是隐式的模型. 包括隐式的映射、通信和同步.	
动态结构	• *Dataflow:* Sisal, Id • *Explicit Logic Languages:* Concurrent Prolog, PARLOG, GHC, Delta-Prolog, Strand • *Multilisp*
静态结构	• *Data Parallelism Using Loops:* Fortran variants, Modula 3* • *Data Parallelism on Types:* pSETL, parallel sets, match and move, Gamma, PEI, APL, MOA, Nial and AT
静态和通信受限的结构	• *Data-Specific Skeletons:* scan, multiprefix, paralations, dataparallel C, NESL, CamlFlight

依据抽象程度进行分类

3 模型和语言的现状和分类

显式地表达并行和分解，但映射、通信和同步是隐式的模型	
静态结构	• BSP, LogP
显式地表达并行、分解和映射，但通信和同步是隐式的模型	
动态结构	• *Coordination Languages:* Linda, SDL • *Non-message Communication Languages:* ALMS, PCN, Compositional C++ • *Virtual Shared Memory* • *Annotated Functional Languages:* Paralf • *RPC:* DP, Cedar, Concurrent CLU, DP
静态结构	• *Graphical Languages:* Enterprise, Parsec, Code • *Contrxtual Coordination Languages:* Ease, ISETL-Linda, Opus
静态和通信受限的结构	• *Communication Skeletons*

3 模型和语言的现状和分类

显式地表达并行、分解、映射和通信，但同步是隐式的模型	
动态结构	• *Process Networks:* Actors, Concurrent Aggregates, ActorSpace, Darwin • *External OO:* ABCL/1, ABCL/R, POOL-T, EPL, Emerald, Concurrent Smalltalk • *Objects and Processes:* Argus, Presto, Nexus • *Active Messages:* Movie
静态结构	• *Process Networks:* static dataflow • *Internal OO:* Mentat
静态和通信受限的结构	• *Systolic Arrays:* Alpha

3 模型和语言的现状和分类

显式地表达所有的东西	
动态结构	· *Message Passing:* PVM, MPI · *Shared Memory:* FORK, Java, thread packages · *Rendezvous:* Ada, SR, Concurrent C
静态结构	· Occam
纯理论模型： PRAM	

3 模型和语言的现状和分类

基于程序构造的模型

- CSP
- Linda

基于问题描述的模型

- GAMMA
- UNITY

基于并行计算理论的模型

- PRAM
- BSP
- LogP

4　模型和语言的评价标准

- 尽可能抽象和简单，易于学习和理解
  隐藏以下编程细节
  - 程序到并行线程的分解
  - 线程到处理器的映射
  - 线程间的通信
  - 线程间的同步
- 体系结构独立
- 能提供一套完整的软件开发方法
- 能够保障性能
- 可测量程序的成本

5 并行性问题

5.1 进程的同构性

- SIMD: 所有进程在同一时间执行相同的指令
- MIMD:各个进程在同一时间可以执行不同的指令
  - SPMD: 各个进程是同构的，多个进程对不同的数据执行相同的代码(一般是数据并行的同义语)
    常对应并行循环，数据并行结构，单代码
  - MPMD:各个进程是异构的， 多个进程执行不同的代码（一般是任务并行，或功能并行，或控制并行的同义语）
    常对应并行块，多代码

要为有1000个处理器的计算机编写一个完全异构的并行程序是很困难的

5 并行性问题

并行块

parbegin S1 S2 S3 …….Sn parend
S1 S2 S3 …….Sn可以是不同的代码

并行循环: 当并行块中所有进程共享相同代码时

parbegin S1 S2 S3 …….Sn parend
S1 S2 S3 …….Sn是相同代码

简化为

parfor (i=1; i<=n, i++) S(i)

5 并行性问题

SPMD程序的构造方法

用单代码方法说明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();
for  (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
```

5 并行性问题

MPMD程序的构造方法

用多代码方法说明MPMD

对不提供并行块或并行循环的语言
要说明以下MPMD程序:
parbegin S1 S2 S3 parend

用户需写3个程序, 分别编译生成3个可执行程序S1 S2 S3, 用shell脚本将它们加载到3个处理结点上:
run S1 on node1
run S2 on node2
run S3 on node3

S1, S2和S3是顺序语言程序加上进行交互的库调用.

用SPMD伪造MPMD

要说明以下MPMD程序:
parbegin S1 S2 S3 parend

可以用以下SPMD程序:

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

因此, 对于可扩展并行机来说, 只要支持SPMD就足够了

5 并行性问题

5.2 静态和动态并行性

程序的结构: 由它的组成部分构成程序的方法

静态并行性: 程序的结构以及进程的个数在运行之前(如编译时, 连接时或加载时)就可确定, 就认为该程序具有静态并行性.

静态并行性的例子:

```
parbegin  P, Q, R  parend

其中P,Q,R是静态的
```

动态并行性: 否则就认为该程序具有动态并行性. 即意味着进程要在运行时创建和终止

动态并行性的例子:

```
while (C>0) begin
        fork (foo(C));
        C:=boo(C);
end
```

5 并行性问题

开发动态并行性的一般方法: Fork/Join

```
Process A:
begin
      Z:=1
      fork(B);
      T:=foo(3);
end
```

```
Process B:
begin
      fork(C);
      X:=foo(Z);
      join(C);
      output(X+Y);
end
```

```
Process C:
begin
      Y:=foo(Z);
end
```

Fork: 派生一个子进程

Join: 强制父进程等待子进程

5 并行性问题

5.3 进程编组

目的：支持进程间的交互，常把需要交互的进程调度在同一组中

一个进程组成员由：组标识符+ 成员序号 唯一确定.

5.4 划分与分配

原则：使系统大部分时间忙于计算，而不是闲置或忙于交互；同时不牺牲并行性(度).

划分：切割数据和工作负载

分配：将划分好的数据和工作负载映射到计算结点(处理器)上

分配方式

显式分配：由用户指定数据和负载如何加载

隐式分配：由编译器和运行时支持系统决定

就近分配原则：进程所需的数据靠近使用它的进程代码

5 并行性问题

并行度(Degree of Parallelism, DOP):同时执行的分进程数.

并行粒度(Granularity): 两次并行或交互操作之间所执行的计算负载.

- 指令级并行
- 块级并行
- 进程级并行
- 任务级并行

并行度与并行粒度大小常互为倒数: 增大粒度会减小并行度.
增加并行度会增加系统(同步)开销

6 交互/通信问题

交互：进程间的相互影响

6.1 交互的类型

- 通信：两个或多个进程间传送数的操作

  通信方式：

  - 共享变量
  - 父进程传给子进程(参数传递方式)
  - 消息传递

6 交互/通信问题

❖同步：导致进程间相互等待或继续执行的操作

同步方式:

- 原子同步
- 控制同步(路障,临界区)
- 数据同步(锁,条件临界区,监控程序,事件)

例子:

原子同步

```
parfor (i:=1; i<n; i++) {
        atomic{x:=x+1; y:=y-1}
        }
```

路障同步

```
parfor(i:=1; i<n; i++){
        P_i
        barrier
        Q_i
        }
```

临界区

```
parfor(i:=1; i<n; i++){
        critical{x:=x+1; y:=y+1}
        }
```

数据同步(信号量同步)

```
parfor(i:=1; i<n; i++){
        lock(S);
        x:=x+1;
        y:=y-1;
        unlock(S)
        }
```

6 交互/通信问题

❖聚集(aggregation)：用一串超步将各分进程计算所得的部分结果合并为一个完整的结果，每个超步包含一个短的计算和一个简单的通信或/和同步.

聚集方式:

- 归约
- 扫描

例子：计算两个向量的内积

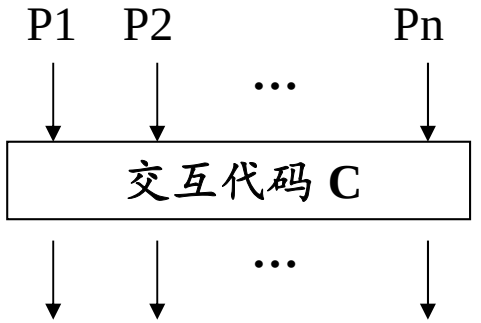
```
parfor(i:=1; i<n; i++){
      X[i]:=A[i]*B[i]
      inner_product:=aggregate_sum(X[i]);
      }
```

6 交互/通信问题

6.2 交互的方式

同步的交互: 所有参与者同时到达并执行交互代码C

异步的交互: 进程到达C后, 不必等待其它进程到达即可执行C



相对于交互代码C,可对进程P定义如下状态:

- 到达(arrived): P刚到达C,但还未进入
- 在内(in): P在代码中
- 完成(finished):P刚完成执行代码C,但还未离开
- 在外(out): P不在代码中(未到达或已离开)

6 交互/通信问题

交互方式与入口/出口条件的组合

入口条件		出口条件		交互方式
当事进程状态	其它进程状态	当事进程状态	其它进程状态	
到达(arrived)	到达(arrived)	完成(finished)	完成(finished)	同步发送/接收
到达(arrived)	×	完成(finished)	×	锁定发送
到达(arrived)	×	完成(finished)	完成(finished)	锁定接收
到达(arrived)	×	在内(in)	×	非锁定发送/接收
到达(arrived)	到达(arrived)	完成(finished)	在内(in)或完成(finished)	路障
到达(arrived)	在外(out)	完成(finished)	在外(out)	临界区

锁定的发送：消息已发完，但不一定已收到
锁定的接收：消息已收到
非锁定的发/收：只是发出发/收的请求

6 交互/通信问题

6.3 交互的模式

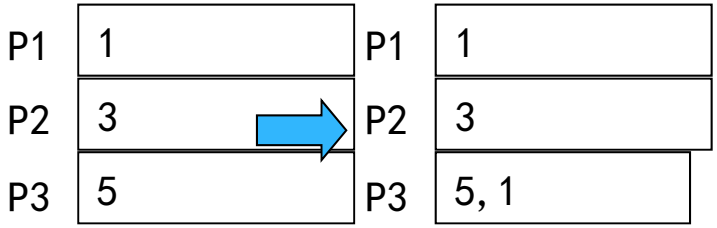
按交互模式是否能在编译时确定分为:

- 静态的
- 动态的

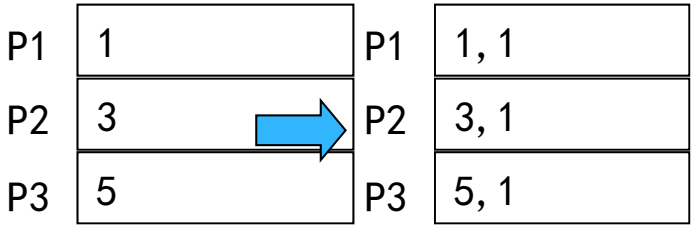
按有多少发送者和接收者参与通信分为

- 一对一:点到点(point to point)
- 一对多:广播(broadcast),播撒(scatter)
- 多对一:收集(gather), 归约(reduce)
- 多对多:全交换(Tatal Exchange), 扫描(scan) , 置换/移位(permutation/shift)

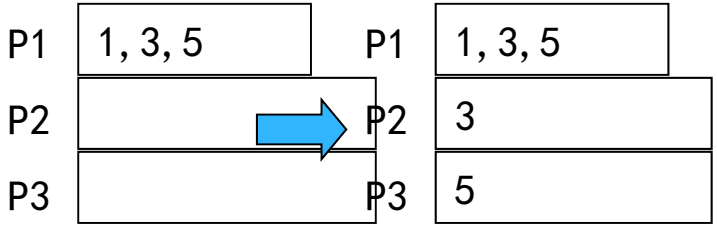
6 交互/通信问题



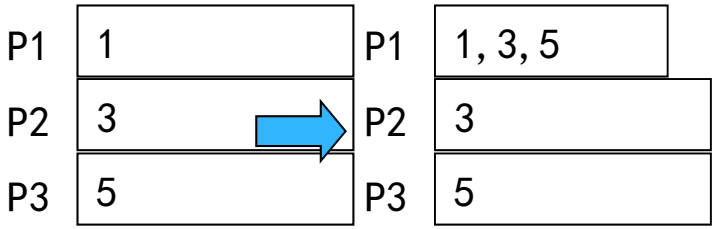
(a) 点对点(一对一)：P1发送一个值给P3



(b) 广播(一对多)：P1发送一个值给全体

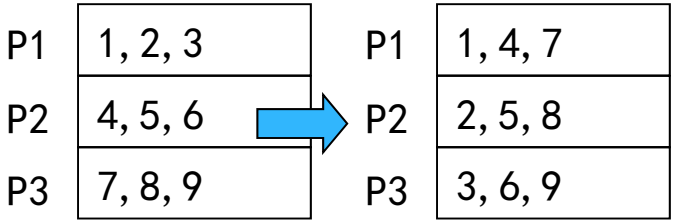


(c) 播撒(一对多)：P1向每个结点发送一个值

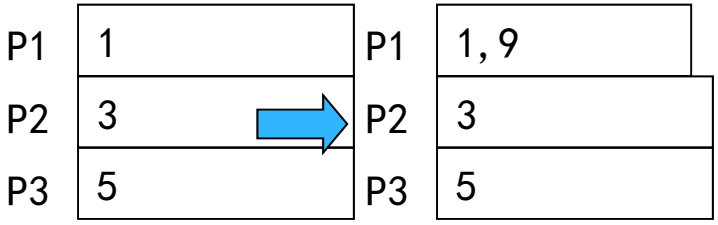


(d) 收集(多对一)：P1从每个结点接收一个值

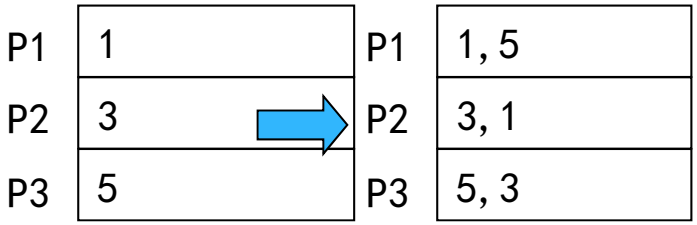
6 交互/通信问题



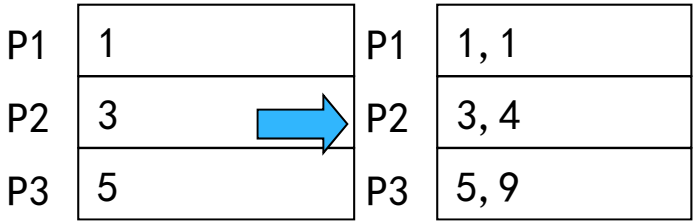
(e) 全交换(多对多)：每个结点向每个结点发送一个不同的消息



(g) 归约(多对一)：P1得到和1+3+5=9



(f) 移位(置换，多对多)：每个结点向下一个结点发送一个值并接收来自上一个结点的一个值.



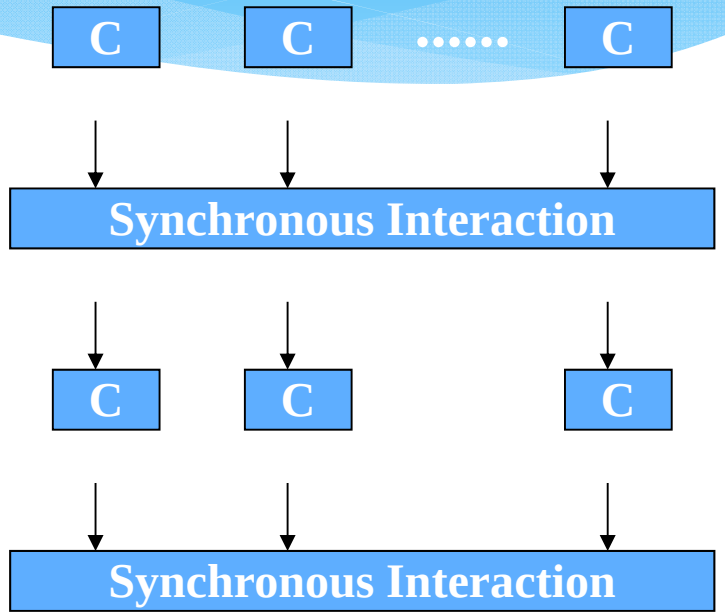
(h) 扫描(多对多)：P1得到1，P2得到1+3=4，P3得到1+3+5=9

7 五种并行编程风范

* 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）

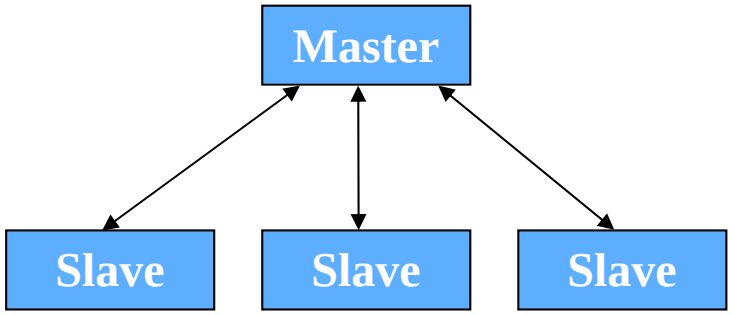
相并行（Phase Parallel）

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



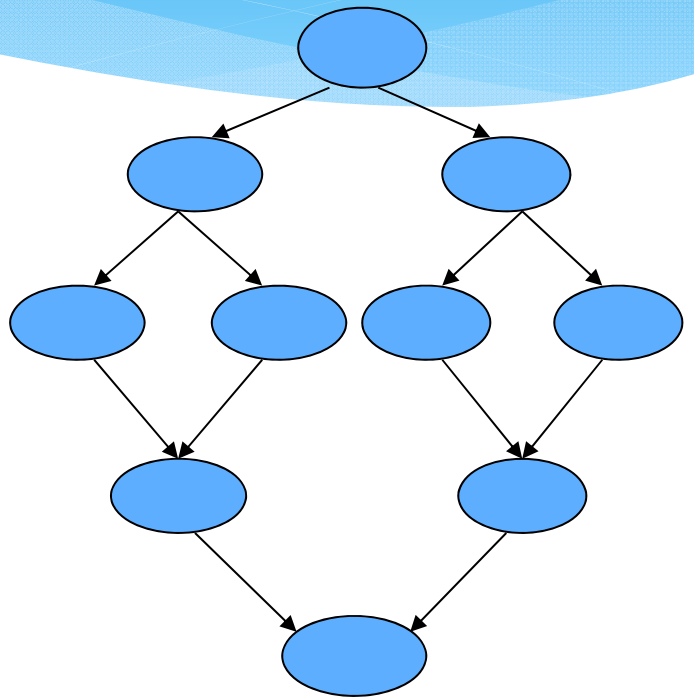
主－从并行（Master-Slave Parallel）

* 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



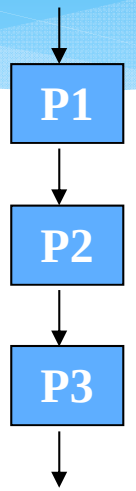
分治并行（Divide and Conquer Parallel）

* 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



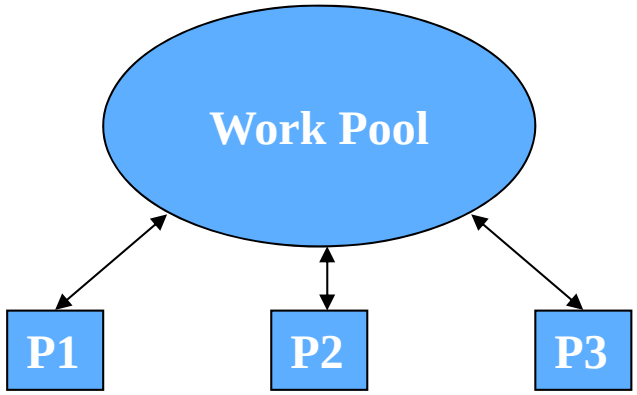
流水线并行（Pipeline Parallel）

* 一组进程
* 流水线作业
* 流水线设计技术（6.5）



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
* 易与负载平衡
* 临界区问题（尤其消息传递）



计算圆周率的C语言代码段

```
#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);
}
```

并行程序设计基础

* 12.1 并行程序设计概述
* 12.2 进程
* 12.3 线程
* 12.4 同步
* 12.5 通信
* 12.6 并行程序设计模型

进程

1 进程的基本概念
2 进程的并行执行
3 进程的相互作用

线程

1 线程的基本概念
2 线程的管理
3 线程的同步

同步

1 原子和互斥
2 高级同步结构
3 低级同步原语

通信

1 影响通信系统性能的因素
2 低级通信支持
3 TCP/IP通信协议组简介

并行程序设计模型

1 隐式并行模型
2 数据并行模型
3 消息传递模型
4 共享变量模型

计算圆周率的样本程序

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

```
#define N 1000000
main(){
double local, pi=0.0, w;
long i;
w=1/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);
}
```

其中，N是等分间隔数，其值越大越精确，但计算时间越长。

并行程序设计模型

* 隐式并行（Implicit Parallel）
* 数据并行（Data Parallel）
* 共享变量（Shared Variable）
* 消息传递（Message Passing）

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
  * 局部计算和数据选路操作
* 特点：
  * 单线程
  * 并行操作于聚合数据结构（数组）
  * 松散同步
  * 单一地址空间
  * 隐式交互作用
  * 显式数据分布

计算圆周率（数据并行）

```
main(){
long N= 1000000
double local[N], temp[N], pi, w;
long i, j, t;
A: w=1/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);
}
```

消息传递（Message Passing）

* 概况：
  * MPP, COW的自然模型
* 特点：
  * 多线程
  * 异步
  * 多地址空间
  * 显式同步
  * 显式数据映射和负载分配
  * 显式通信

计算圆周率（消息传递）

```
#define N 1000000
main(){
double local, pi, w;
long i, taskid, numtask;
A: w=1/N;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
MPI_Comm_size(MPI_COMM_WORLD, & numtask);
```

计算圆周率（消息传递）

B: **for**(i=taskid;i<N;i=i+numtask){

P: local=(i+0.5)*w;

Q: local=4.0/(1.0+local*local);

}

C: MPI_Reduce(&local, &pi,1,MPI_Double, MPI_MAX,0,MPI_COMM_WORLD);

D: if(taskid==0) printf(“pi is %f\n”,pi*w);

MPI_Finalize();

}

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
  * 单一地址空间
  * 显式同步
  * 隐式数据分布
  * 隐式通信

计算圆周率（共享变量）

```
#define N 1000000
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 iterator(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);
}
```

并行程序设计模型比较

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