并行计算
Parallel Computing

主讲 孙经纬

2024年 春季学期

概要

- 第二篇 并行算法的设计
  - **第五章 并行算法与并行计算模型**
  - 第六章 并行算法基本设计策略
  - 第七章 并行算法常用设计技术
  - 第八章 并行算法一般设计过程

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
  - **5.1.1 并行算法的定义和分类**
  - 5.1.2 并行算法的表达
  - 5.1.3 并行算法的复杂性度量
  - 5.1.4 并行算法中的同步和通讯
- 5.2 并行计算模型

并行算法的定义和分类

- 并行算法的定义
  - 算法：略
  - 并行算法：一些可同时执行的多个进程（线程/处理器/节点…… Anyway）的集合，这些进程互相作用和协调动作从而达到给定问题的求解。
- 并行算法的分类
  - 数值计算和非数值计算
  - 同步算法和异步算法
  - 分布算法
  - 确定算法和随机算法

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
  - 5.1.1 并行算法的定义和分类
  - **5.1.2 并行算法的表达**
  - 5.1.3 并行算法的复杂性度量
  - 5.1.4 并行算法中的同步和通讯
- 5.2 并行计算模型

并行算法的表达

- 描述语言
  - 可以使用各类语言风格的伪代码;
  - 在描述语言中引入并行语句。
- 并行语句示例
  - Par-do语句

```
for i=1 to n par-do
        .....
end for
```

  - for all语句

```
for all P_i, where 0≤i≤k  do
    ......
end for
```

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
  - 5.1.1 并行算法的定义和分类
  - 5.1.2 并行算法的表达
  - **5.1.3 并行算法的复杂性度量**
  - 5.1.4 并行算法中的同步和通讯
- 5.2 并行计算模型

并行算法的复杂性度量

- 串行算法的复杂性度量
  - 最坏情况下的复杂度(Worst-Case Complexity)
  - 期望复杂度(Expected Complexity)

并行算法的复杂性度量

- 并行算法的几个复杂性度量指标
  - 运行时间t(n): 包含计算时间和通讯时间，分别用计算时间步和选路时间步作单位。n为问题实例的输入规模。
  - 处理器数p(n)
  - 并行算法成本c(n): c(n)=t(n)p(n)
  - 成本最优性：若c(n)等于在最坏情形下串行算法所需要的时间，则并行算法是成本最优的。
  - 总运算量W(n): 并行算法求解问题时所完成的总的操作步数。

并行算法的复杂性度量

- Brent定理

令W(n)是某并行算法A在运行时间T(n)内所执行的运算量，则A使用p个处理器可在

$$t(n)=O(W(n)/p+T(n))$$

时间内执行完毕。

Brent定理一种更常见的表述形式是：

$$t(n)<=T(n)+(W(n)-T(n))/p$$

并行算法的复杂性度量

- Brent定理证明

设w_j是算法A在第j时间步的可并行运算量，j=1, 2, … T(n)，$w(n) = \sum_{j=1}^{T(n)} w_j$。p应当不大于max($w_j$)，并且$\left\lceil \frac{w_j}{p} \right\rceil \le \frac{w_j+p-1}{p}$。

上取整函数的基本性质

则A使用p个处理器执行W(n)运算量的耗时t(n)满足：

$$t(n) \le \sum_{j=1}^{T(n)} \left\lceil \frac{w_j}{p} \right\rceil \le \sum_{j=1}^{T(n)} \frac{w_j + p - 1}{p} = T(n) - \frac{T(n)}{p} + \sum_{j=1}^{T(n)} \frac{w_j}{p}$$

$$t(n) \le T(n) + \frac{W(n) - T(n)}{p} \Rightarrow t(n) = O(T(n) + \frac{W(n)}{p})$$

并行算法的复杂性度量

- Brent定理的意义
  - 提供了并行算法的WT(Work-Time)表示方法；也可称之为Work-Depth模型。
  - 由于数据依赖等原因，并行算法中必然存在至少一条串行执行的运算操作序列。其中最长的一条叫做关键路径(critical path)。
  - 设计并行算法时应尽可能将每个时间步的工作量均匀地分摊给p台处理器，使各处理器处于活跃状态。这要求关键路径的长度尽可能的短。

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
  - 5.1.1 并行算法的定义和分类
  - 5.1.2 并行算法的表达
  - 5.1.3 并行算法的复杂性度量
  - **5.1.4 并行算法中的同步和通讯**
- 5.2 并行计算模型

并行算法的同步

- 同步概念
  - 同步是在时间上强使各执行进程在某一点必须互相等待;
  - 可用软件和硬件的办法来实现。
- 同步语句示例
  - 共享存储多处理器上求和算法

输入：$A=(a_0,\ldots,a_{n-1})$,处理器数p

输出：$S=\Sigma a_i$

```
Begin
    (1)S=0
    (2)for all P_i where 0≤i≤p-1 do
      (2.1) L=0
      (2.2) for j=i to n step p do
                L=L+a_j
            end for
      (2.3) lock(S)
            S=S+L
      (2.4) unlock(S)
      end for
End
```

为什么要step p?

Example: Critical Section

```
#define NUMTHREADS 4
CRITICAL_SECTION g_cs; // why does this have to be global?
int g_sum = 0;

DWORD WINAPI threadFunc(LPVOID arg )
{
  int mySum = bigComputation();
  EnterCriticalSection(&g_cs);
   g_sum += mySum;                  // threads access one at a time
  LeaveCriticalSection(&g_cs);
  return 0;
}
main() {
  HANDLE hThread[NUMTHREADS];
  InitializeCriticalSection(&g_cs);
  for (int i = 0; i < NUMTHREADS; i++)
    hThread[i] =
        CreateThread(NULL,0,threadFunc,NULL,0,NULL);
  WaitForMultipleObjects(NUMTHREADS, hThread, TRUE, INFINITE);
  DeleteCriticalSection(&g_cs);
}
```

并行算法的通讯

- 通讯
  - 共享存储多处理器使用：global read(X,Y)和global write(X,Y)
  - 分布存储多计算机使用：send(X,i)和receive(Y,j)
- 通讯语句示例
  - 算法5.2 分布存储多计算机上矩阵向量乘算法

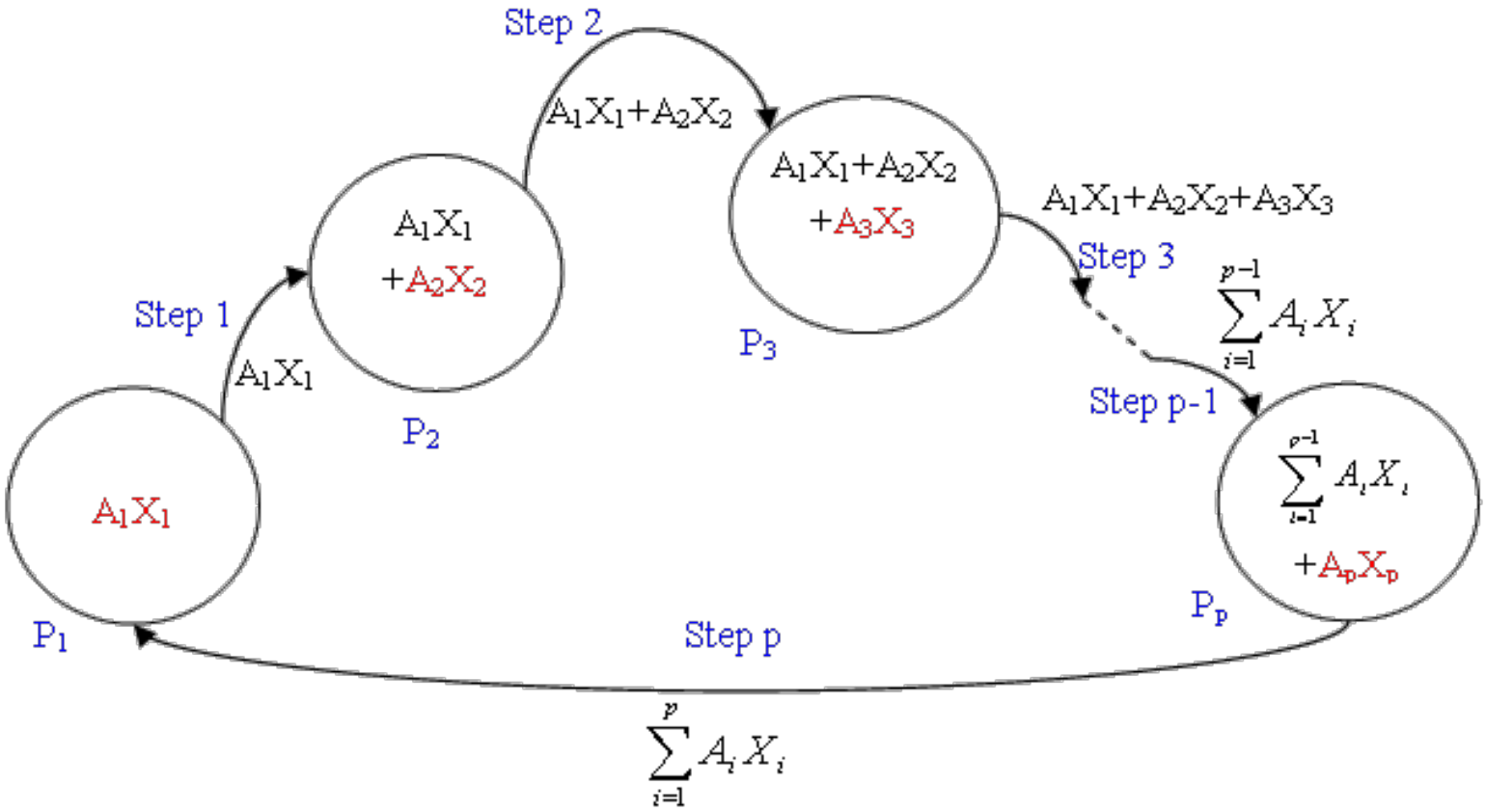
输入：处理器数p, A划分为B=A[1..n,(i-1)r+1..ir],

X划分为w=w[(i-1)r+1..ir]　r=n/p, i=1~p

输出：P_1保存乘积AX

```
Begin
    (1) Compute z=Bw
    (2) if i==1 then y=0 else receive(y,left) endif
    (3) y=y+z
    (4) send(y,right)
    (5) if i==1 then receive(y,left)
End
```

并行算法的通讯


Step 2
$A_1X_1+A_2X_2$
A_1X_1
$+A_2X_2$
P_2
$A_1X_1+A_2X_2$
$+A_3X_3$
P_3
$A_1X_1+A_2X_2+A_3X_3$
Step 3
$\sum_{i=1}^{p-1} A_i X_i$
Step p-1
$\sum_{i=1}^{p-1} A_i X_i$
$+A_pX_p$
P_p
Step 1
A_1X_1
A_1X_1
P_1
Step p
$\sum_{i=1}^{p} A_i X_i$

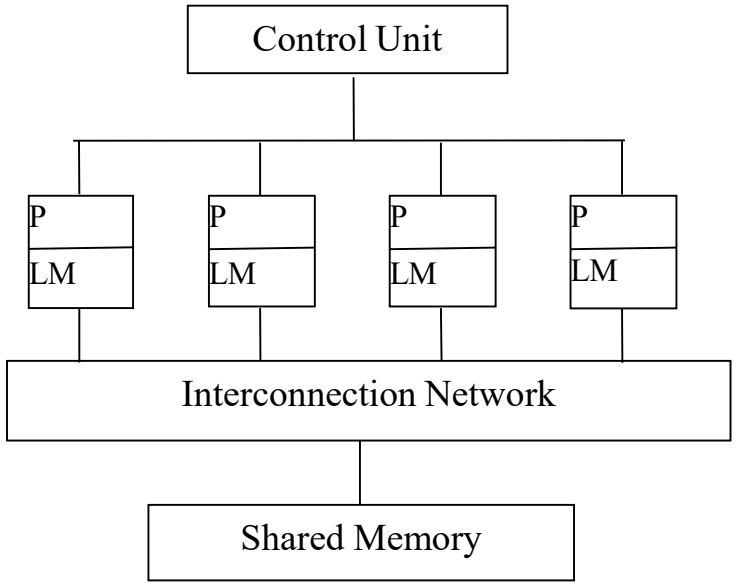
第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
- 5.2 并行计算模型
  - **5.2.1 PRAM模型**
  - 5.2.2 异步PRAM模型
  - 5.2.3 BSP模型
  - 5.2.4 logP模型

PRAM模型

- Parallel Random Access Machine
- 基本概念
  - 由Fortune和Wyllie1978年提出，又称**SIMD-SM**模型。有一个集中的共享存储器和一个指令控制器，通过共享存储器的读写交换数据，隐式同步计算。
- 结构图

单指令多数据-共享存储



PRAM模型

- PRAM-CRCW并发读并发写

  CPRAM-CRCW(Common PRAM-CRCW)：仅允许写入相同数据

  PPRAM-CRCW(Priority PRAM-CRCW)：仅允许优先级最高的处理器写入

  APRAM-CRCW(Arbitrary PRAM-CRCW)：允许任意处理器自由写入

- PRAM-CREW并发读互斥写
- PRAM-EREW互斥读互斥写
- 计算能力比较
  - PRAM-CRCW是最强的计算模型，PRAM-EREW可logp倍模拟PRAM-CREW和PRAM-CRCW

$$T_{EREW} \geq T_{CREW} \geq T_{CRCW}$$

$$T_{EREW} = O(T_{CREW} \cdot \log p) = O(T_{CRCW} \cdot \log p)$$

ER模拟CR：树形传播
EW模拟CW：树形竞赛

PRAM模型

- 优点
  - 适合并行算法表示和复杂性分析，易于使用
  - 隐藏了并行机的通讯、同步等细节
  - 易于设计算法，适量修改可运行在实际的并行机上
    - OpenMP (共享内存多核CPU上多线程并行程序)
    - CUDA (GPU上的单指令多线程并行程序)

- 缺点
  - 不适合MIMD并行机
  - 假定每个处理器可在单位时间内访问任何存储单元
  - 忽略了共享存储器的竞争、通讯延迟等因素

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
- 5.2 并行计算模型
  - 5.2.1 PRAM模型
  - **5.2.2 异步PRAM模型**
  - 5.2.3 BSP模型
  - 5.2.4 logP模型

APRAM模型

- Asynchronous PRAM
- 基本概念
  - 又称分相（Phase）PRAM或MIMD-SM。每个处理器有其局部存储器、局部时钟、局部程序；无全局时钟，各处理器异步执行；处理器通过SM进行通讯；处理器间依赖关系，需在并行程序中显式地加入同步路障。
- 指令类型

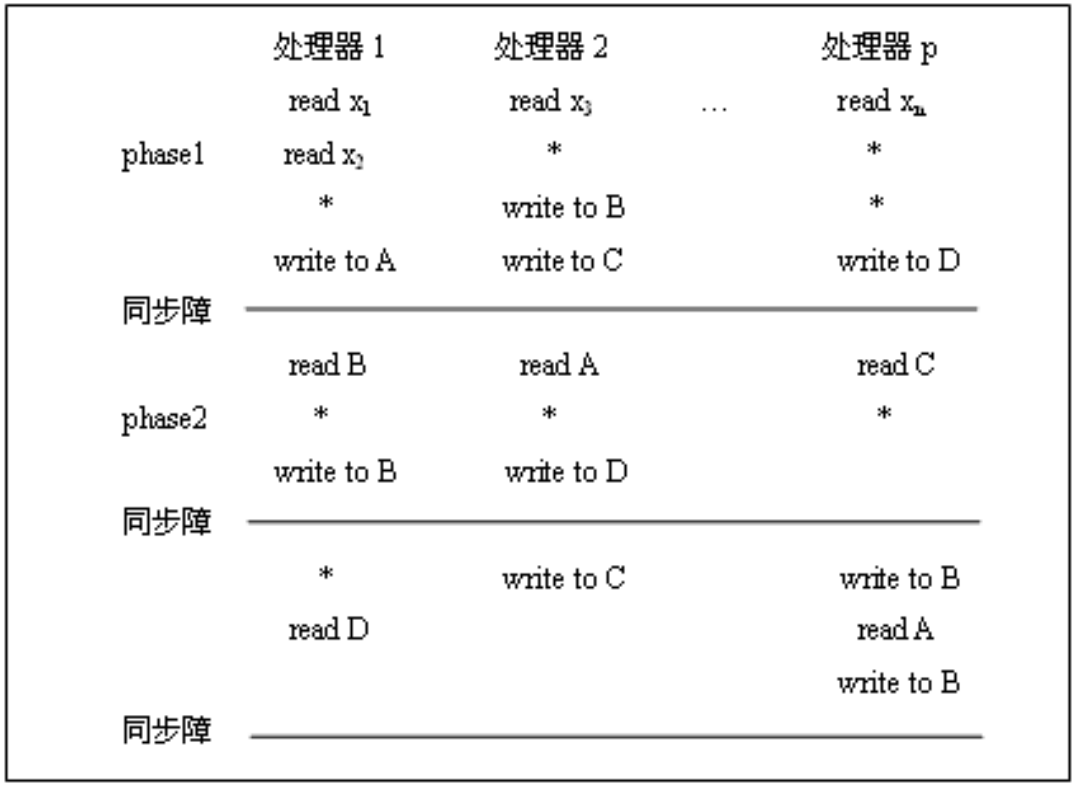
  (1)全局读　　(2)全局写

  (3)局部操作　　(4)同步

多指令多数据-共享存储

APRAM模型

- 计算过程
  由同步障分开的全局相组成



APRAM模型

相比于PRAM，增加了MIMD表达能力，并且量化了通信开销

- 计算时间

  设局部操作为单位时间；全局读/写平均时间为d，d随着处理器数目的增加而增加；同步路障时间为B=B(p)非降函数。

  满足关系 $2 \le d \le B \le p$；$B = B(p) = O(d \log p)$或 $O(d \log p / \log d)$

  令 t_{ph} 为全局相内各处理器执行时间最长者，则APRAM上的计算时间为

$$T = \sum t_{ph} + B \times \text{同步障次数}$$

- 优缺点

  易编程和分析算法的复杂度，但与现实机器相差较远，其上并行算法非常有限，不适合MIMD-DM。

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
- 5.2 并行计算模型
  - 5.2.1 PRAM模型
  - 5.2.2 异步PRAM模型
  - **5.2.3 BSP模型**
  - 5.2.4 logP模型

BSP模型

“大”同步并行
相应地，APRAM是“小”同步并行

- Bulk Synchronous Parallel
- 基本概念
  - 由Leslie Valiant(1990)提出的，“块”同步模型，是一种**异步MIMD-DM**模型，支持消息传递系统，块内异步并行，块间显式同步。
- 模型参数
  - *p*：处理器数(带有存储器)
  - *l*：同步障时间(Barrier synchronization time)
  - *g*：带宽因子(time steps/packet)=1/bandwidth

BSP模型

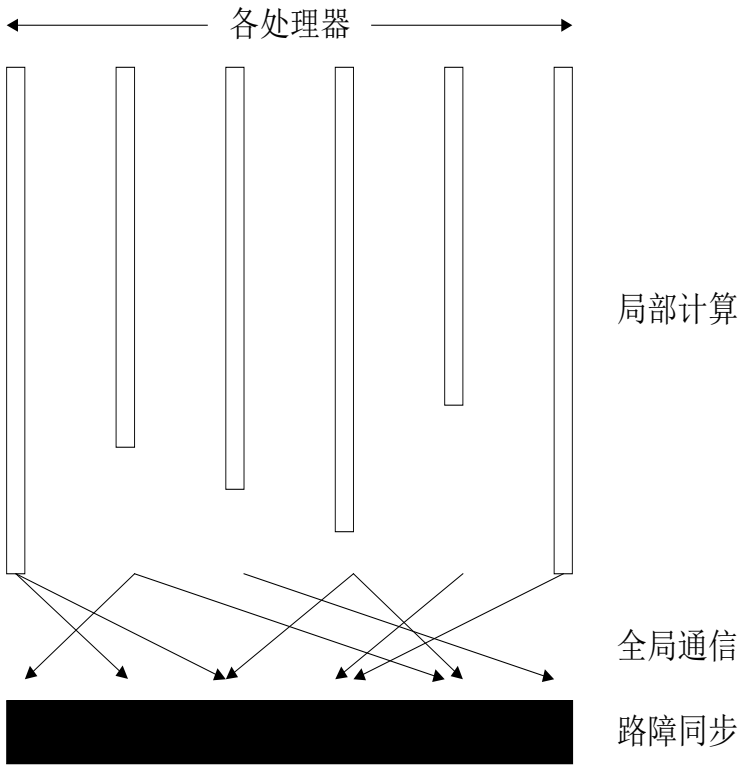
- 计算过程

  由若干超级步组成，

  每个超级步计算模式为右图

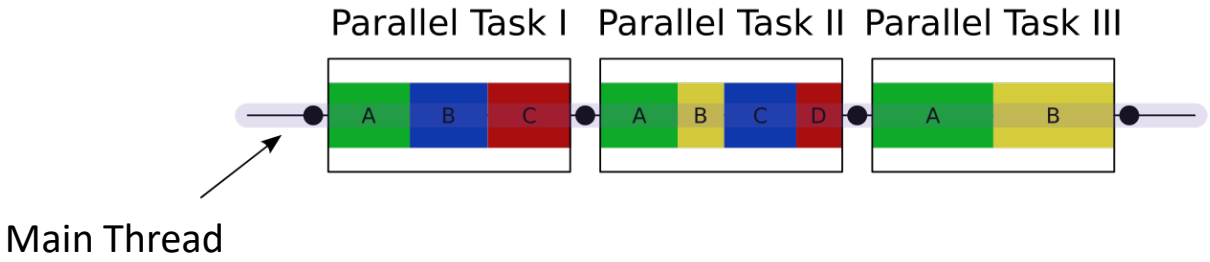
- 优缺点

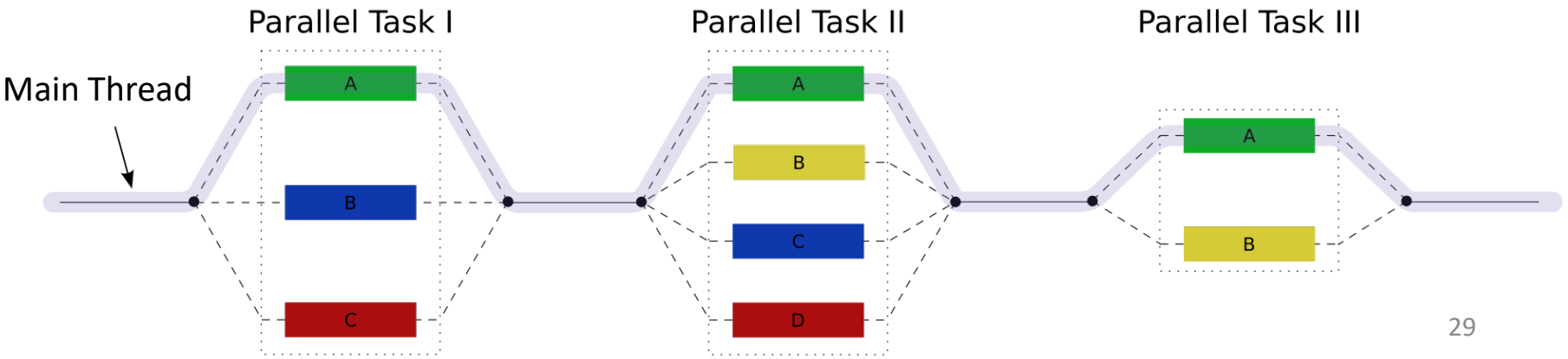
  强调了计算和通讯的分离，提供了一个编程环境，易于程序复杂性分析。但需要显式同步机制，限制至多h条消息的传递等。



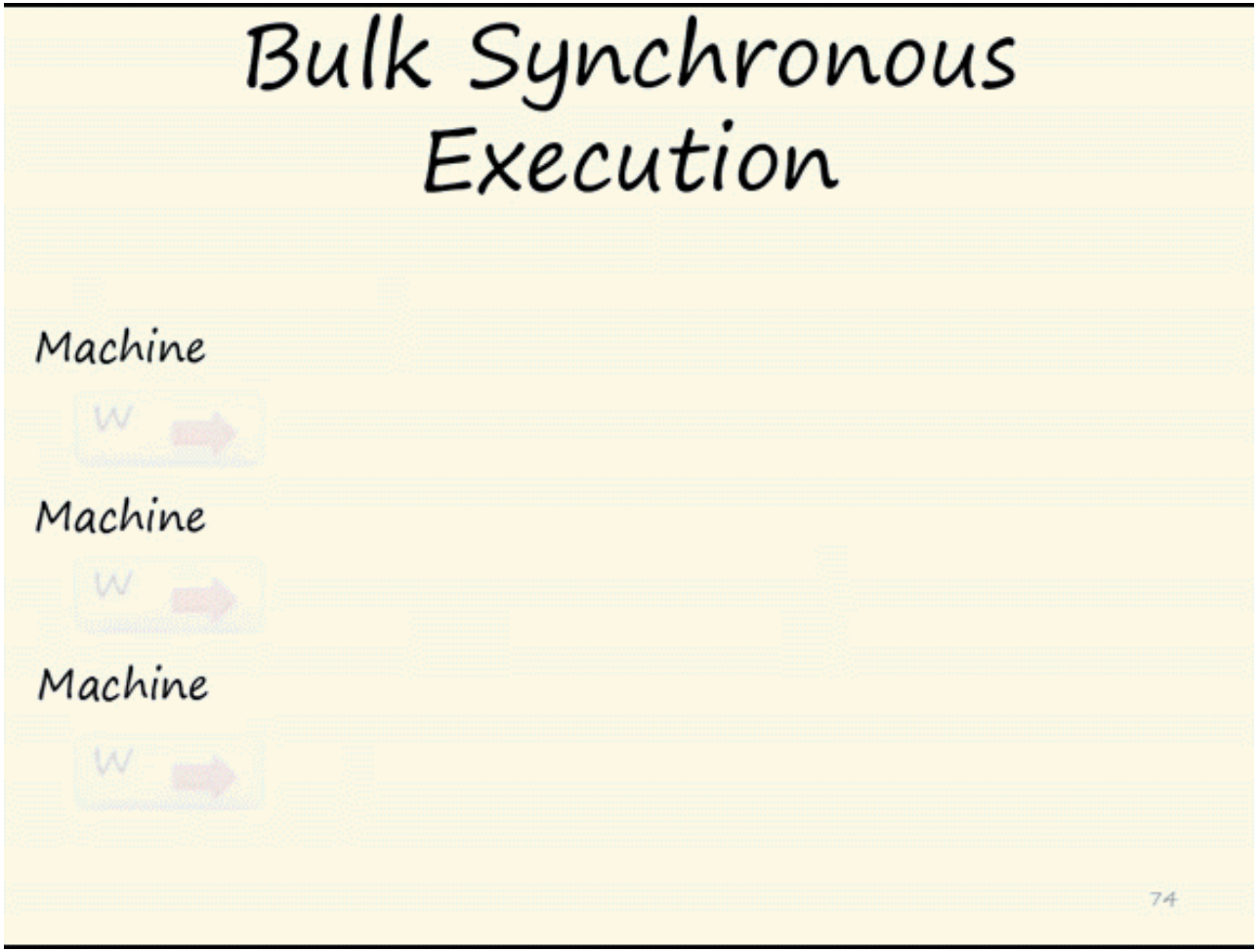
BSP模型

- 在算法设计、编程实现时，符合人的思维直觉
- 基于BSP设计的算法易于转换为实际的MPI程序
- “Fork-Join”是BSP的一个特例

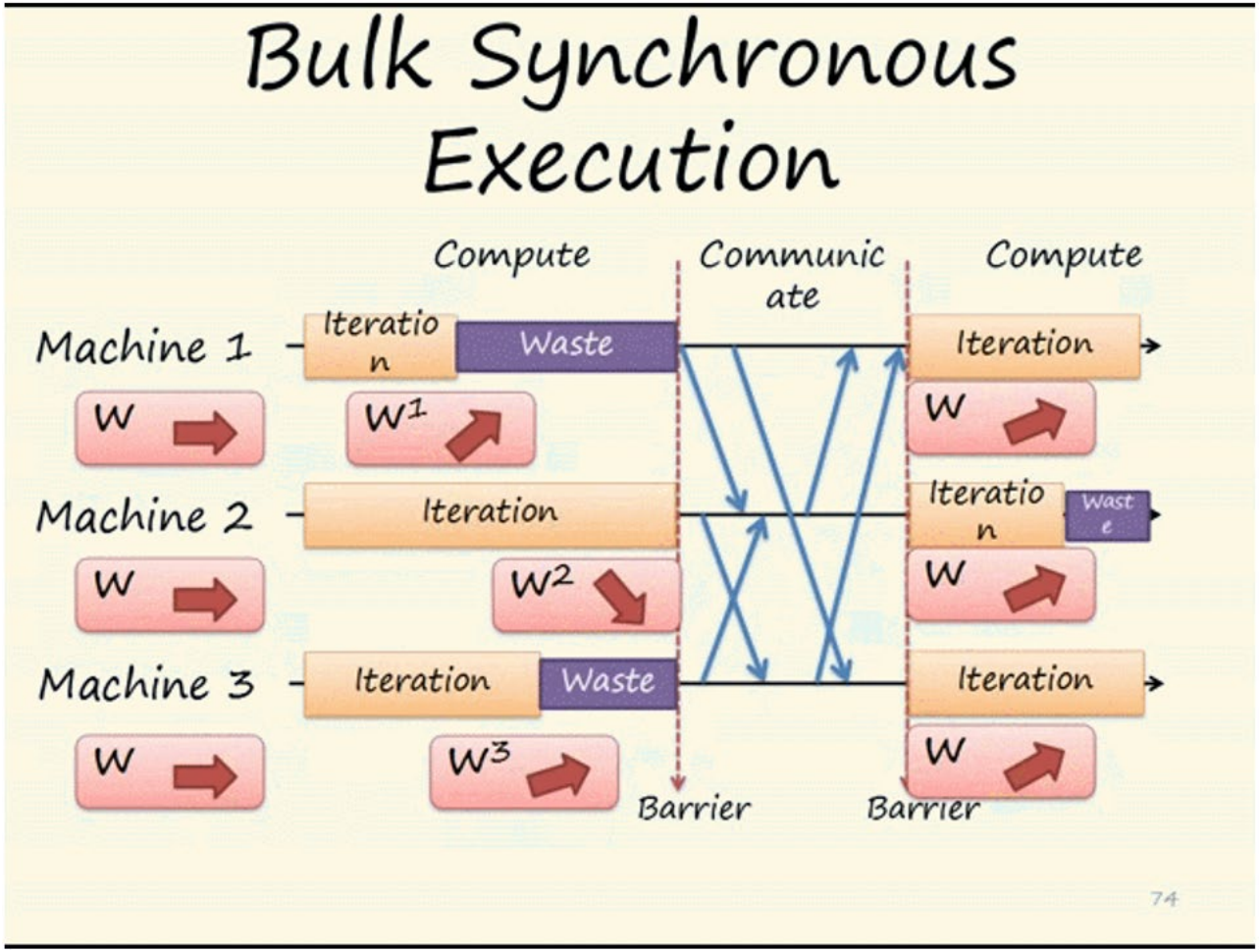




BSP


Bulk Synchronous Execution
Machine
W
Machine
W
Machine
W
74

BSP


Bulk Synchronous Execution
Compute
Communic ate
Compute
Machine 1
Machine 2
Machine 3
Iteratio n
Waste
Iteration
W
W1
W2
W3
Iteratio n
Wast e
Barrier
Barrier
74

第五章 并行算法与并行计算模型

- 5.1 并行算法的基础知识
- 5.2 并行计算模型
  - 5.2.1 PRAM模型
  - 5.2.2 异步PRAM模型
  - 5.2.3 BSP模型
  - **5.2.4 logP模型**

LogP模型

MIMD-DM
相比BSP，没有路障，提供了更细致的通信开销描述

- Latency-overhead-gap-Processor
- 基本概念
  - 由Culler(1993)年提出的，是一种分布存储的、点到点通讯的多处理机模型，其中通讯由一组参数描述，实行隐式同步。
- 模型参数
  - *L*：network latency
  - *o*：communication overhead
  - *g*：gap=1/bandwidth
  - *P*：#processors

注：*L*和*g*反映了通讯网络的容量

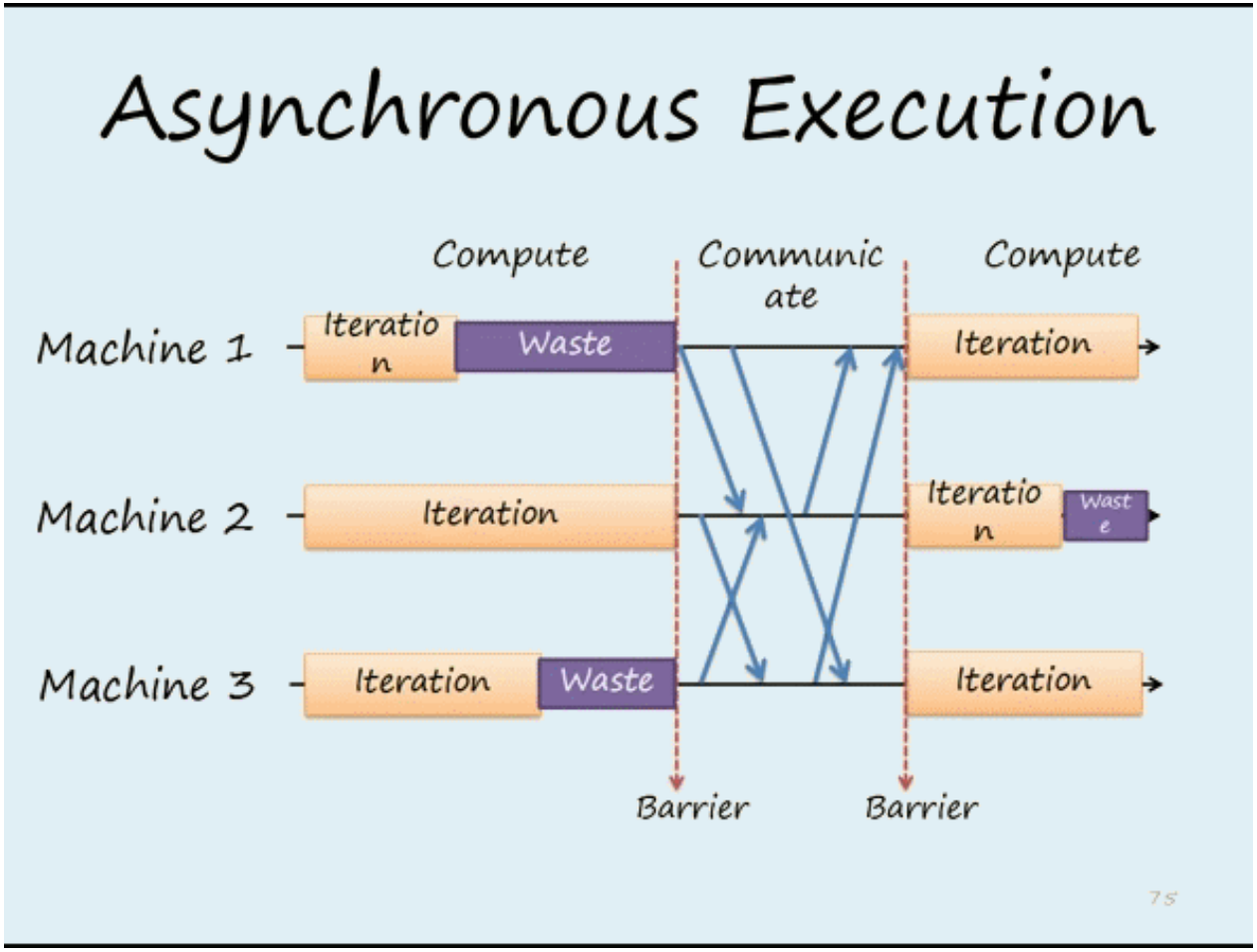
LogP模型

- 优缺点

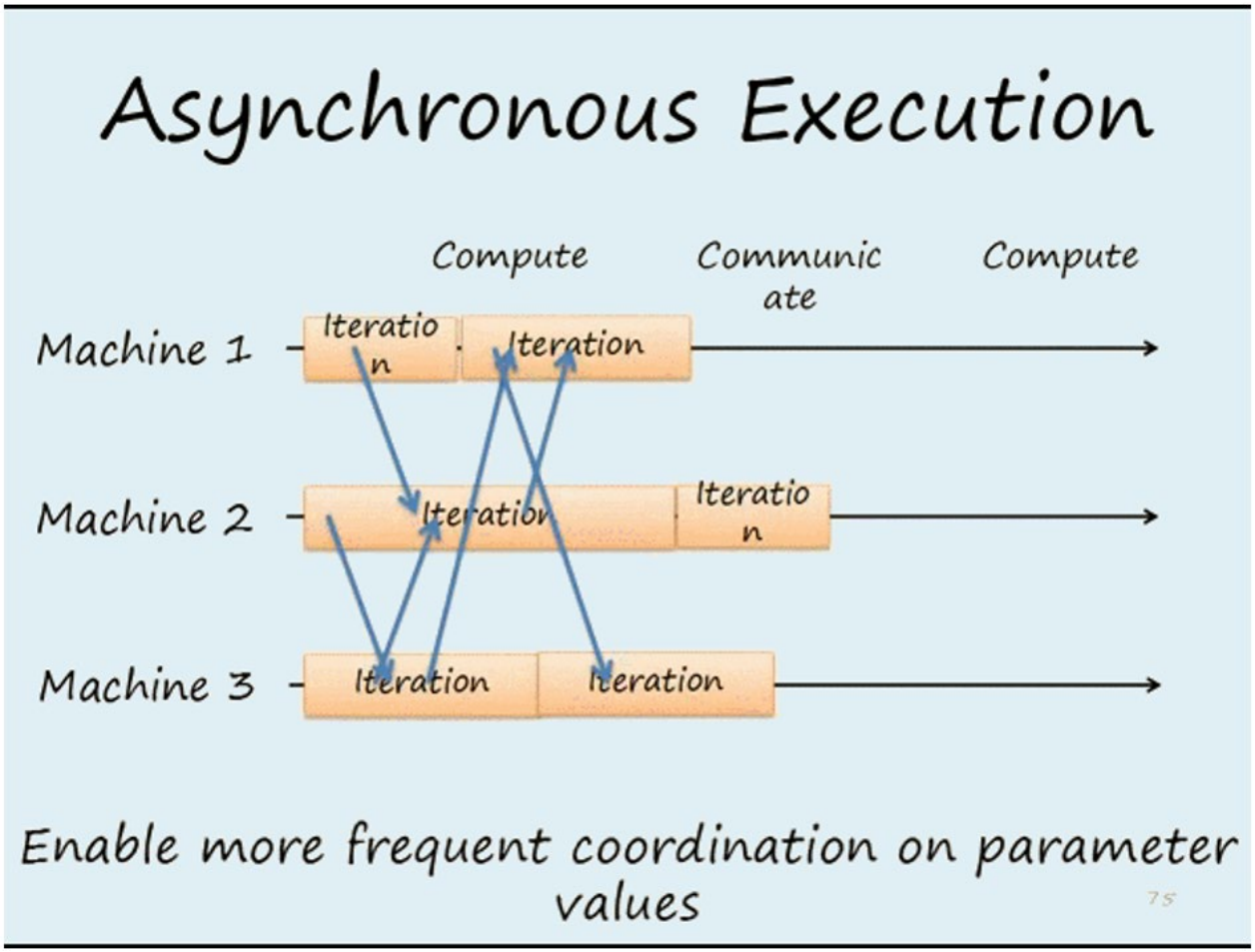
  捕捉了并行机的通讯瓶颈，隐藏了并行机的网络拓扑、路由、协议，可以应用到共享存储、消息传递、数据并行的编程模型中；难以进行算法描述、设计和分析。

- **BSP vs. LogP**
  - BSP→LogP：BSP块同步→BSP子集同步(分组同步)→BSP进程对同步 = LogP
  - BSP可以常数因子模拟LogP，LogP可以对数因子模拟BSP
  - BSP = LogP + Barriers - Overhead
  - BSP提供了更方便的算法设计和编程环境，LogP更好地利用了机器资源
  - BSP更简单、方便和符合结构化编程

(理想中的) LogP


Asynchronous Execution
Compute
Communic ate
Compute
Machine 1
Iteratio n
Waste
Iteration
Machine 2
Iteration
Iteratio n
Wast e
Machine 3
Iteration
Waste
Iteration
Barrier
Barrier
75

(理想中的) LogP


Asynchronous Execution
Compute
Communic ate
Compute
Machine 1
Iteratio n
Iteration
Machine 2
Iteration
Iteratio n
Machine 3
Iteration
Iteration
Enable more frequent coordination on parameter values
75

LogP model family

- LogP
- LogGP
- LogGPS
- LogGOPS
- LogGOPSC
- ……

并行计算模型参考文献

PRAM
- Steven Fortune, James Wyllie: Parallelism in Random Access Machines. STOC 1978: 114-118
- Richard Cole, Ofer Zajicek: The APRAM: Incorporating Asynchrony into the PRAM Model. SPAA 1989: 169-178

BSP
- Leslie G. Valiant: A Bridging Model for Parallel Computation. Commun. ACM 33(8): 103-111 (1990)
- David E. Culler, Richard M. Karp, David A. Patterson, Abhijit Sahay, Klaus E. Schauser, Eunice E. Santos, Ramesh Subramonian, Thorsten von Eicken: LogP: Towards a Realistic Model of Parallel Computation. PPoPP 1993: 1-12
- Albert D. Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, Chris J. Scheiman: LogGP: Incorporating Long Messages into the LogP Model - One Step Closer Towards a Realistic Model for Parallel Computation. SPAA 1995: 95-105
- Fumihiko Ino, Noriyuki Fujimoto, Kenichi Hagihara: LogGPS: a parallel computational model for synchronization analysis. PPoPP 2001: 133-142
- Torsten Hoefler, Timo Schneider, Andrew Lumsdaine: LogGOPSim: simulating large-scale applications in the LogGOPS model. HPDC 2010: 597-604
- Baicheng Yan, Yi Zhou, Limin Xiao, Jiantong Huo, Zhaokai Wang: LogGOPSC: A Parallel Computation Model Extending Network Contention into LogGOPS. CLUSTER 2019: 1-2

One more thing ……

从不同的并行计算机来(抽象计算参数建立模型)，经过不同的加工后，又回到不同的并行计算机中去(运行代码，求解问题)。