Multicore Programming

Outline

- What is parallel computing
- How create a parallel computing

什麼是並行計算
•如何創建並行計

Computation

- **Serial computation**
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another on CPU.
- **Parallel computation**
  - Dividing a problem into discrete parts
  - Each part can be solved concurrently
  - Instructions from each part execute simultaneously on different processing elements

•串行計算
–將問題分解為一系列離散的指令。
–指令在CPU上一個接一個地執行。

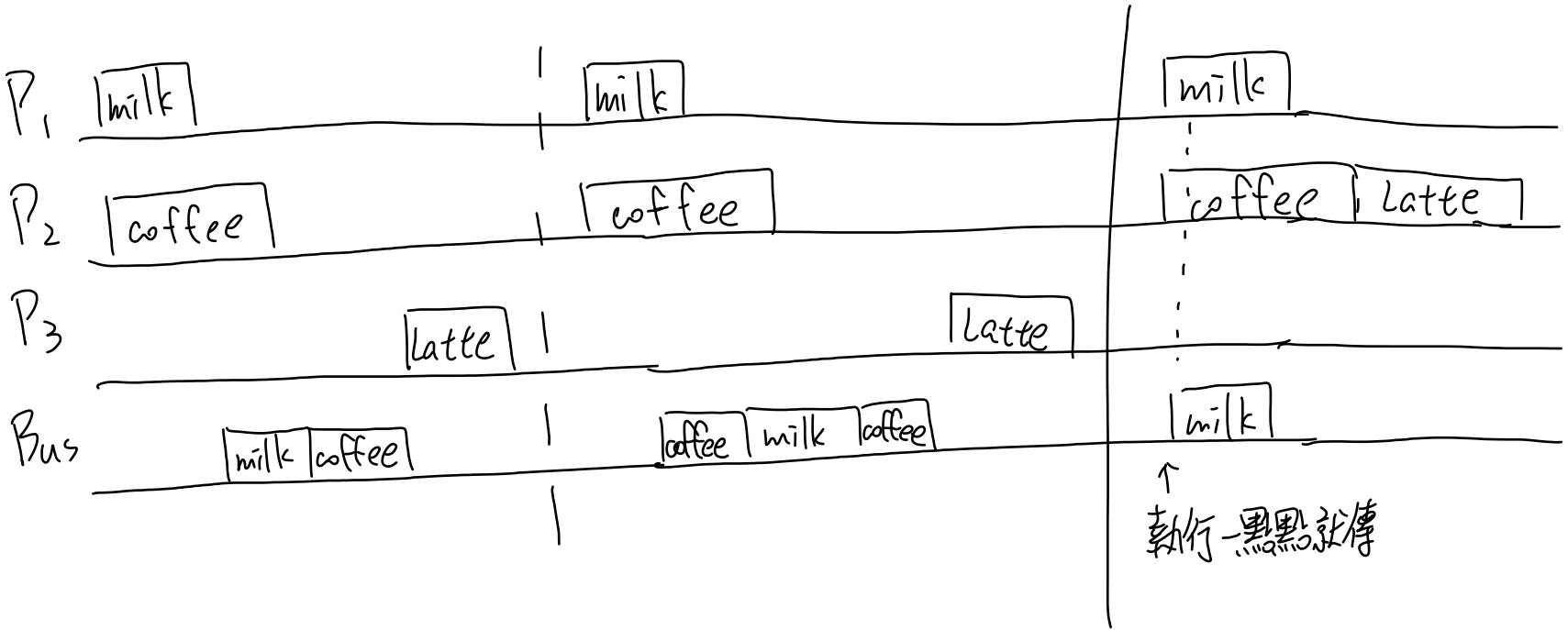
•並行計算
–將問題分為不同的部分
–每個部分可以同時解決
–每個部分的指令在不同的處理元素上同時執行

沒有前後關係可平行

How create a Parallel Program

- Decomposition
- Assignment
- Orchestration
- Mapping

如何創建並行程序
- 分解
- 任務
- 編排
- 映射



Issues with Parallel Computing

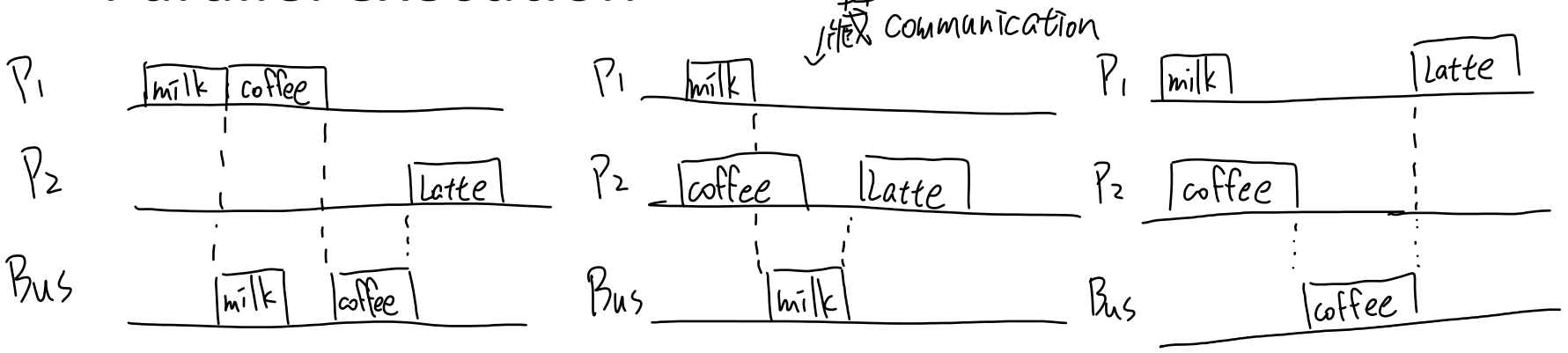
- Problem Decomposition
  - Domain decomposition
  - Functional decomposition
- Data dependency and communication 數據依賴和通訊
- Synchronization 同步化
- Parallel execution 並行執行

並行計算的問題
•問題分解
–域分解
–功能分解
•數據依賴性和通信
•同步
•並行執行

平行

for(i=0; i++; i<10) A[i] = B[i] + C[i] O

for(i=0; i++; i<10) A[i] = A[i-1] + B[i] X



數據/控制並行

Data/ Control Parallelism

pthread_create（
/ *線程ID * /
/* 屬性 */,
/ *任何功能* /,
/ * args功能* /）;

實用程序並行

```
#pragma omp parallel
#pragma omp for
    for(i = 0; i < 12; i++)
        C[i] = A[i] + B[i];
```

東西為

```
pthread_create(
    /* thread id */,
    /* attributes */,
    /* any function */,
    /* args to function */);
```

數據依賴與溝通

Data Dependency and Communication

- When two parts have data dependencies
  - cannot be executed in parallel
  - the order of the operations is critical
    - RAW, WAR, WAW
- When two parts need communication
  - to exchange data
  - to send a message
  - introduce overhead

•當兩個部分具有數據依賴性時
–不能並行執行
–操作順序至關重要
•RAW，WAR，WAW

R= read W= write
A= after

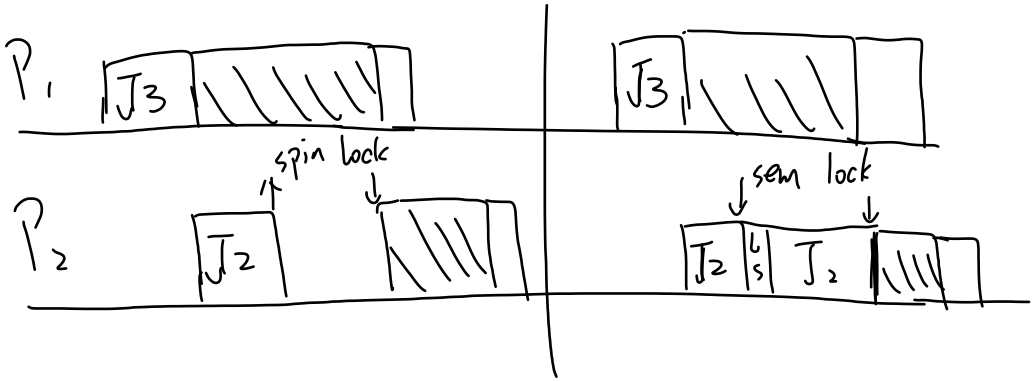
•當兩個部分需要溝通時
–交換數據
–發送消息
–引入開銷

同步化

Synchronization

- Semaphore 可初始為1 or 0
- Mutex pxo 繼承
- Interrupt disabling
- Spin lock 用硬體的lock，不會讓其它人搶走（Bussing waiting）

•信號量
•互斥體
•中斷禁止
•自旋鎖



•單指令流，單數據流（SISD）
•單指令流，多個數據流（SIMD）
•多個指令流，單個數據流（MISD）
•多個指令流，多個數據流（MIMD）
– SPMD（單程序，多數據）
– MPMD（多個程序，多個數據）

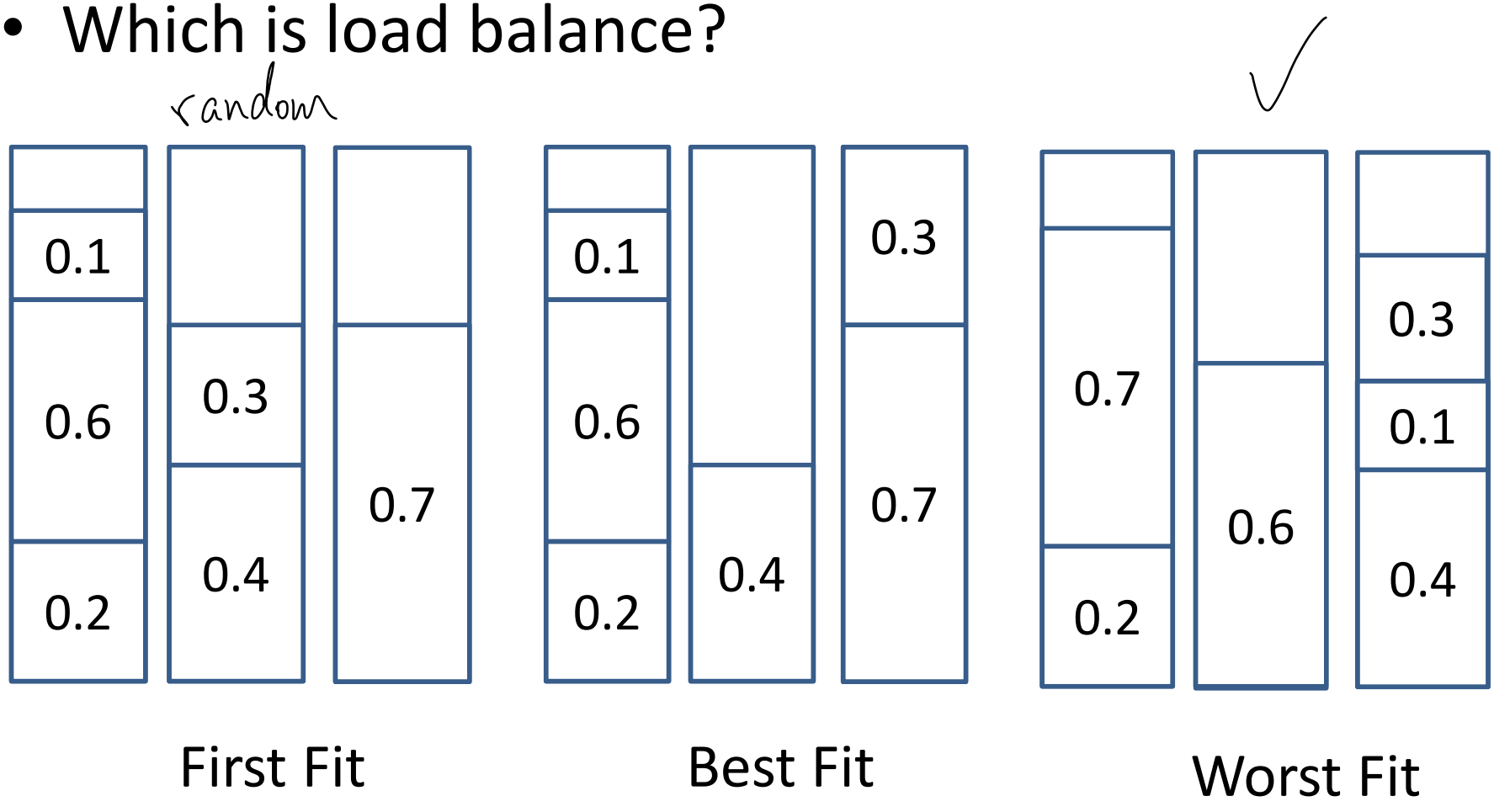
並行執行Flynn的分類法

Parallel execution
Flynn's Taxonomy

- Single Instruction stream, Single Data stream (SISD)
- Single Instruction stream, Multiple Data streams (SIMD)
- Multiple Instruction streams, Single Data stream (MISD)
- Multiple Instruction streams, Multiple Data streams (MIMD)
  - SPMD (Single Program, Multiple Data)
  - MPMD (Multiple Programs, Multiple Data)

Load Balance

- Which is load balance?



靜態負載平衡

Static Load Balancing

- Assigns a fixed amount of work to each core in a prior 事先為每個核心分配固定數量的工作
- Better for homogeneous multicores 更適合均質多核
  - All core are identical –所有核心均相同
  - All the processors have the same characteristics –所有處理器均具有相同的特性
- About heterogeneous multicores
  - Each task has its own execution time on a specified processor
  - A job might be executed faster on a processor, but other jobs might be slower on that processor.
- Examples: Loop, array

示例：循環，數組

關於異構多核
–每個任務在指定處理器上都有自己的執行時間
–作業可能在處理器上執行得更快，但是其他作業在該處理器上執行得可能更慢。

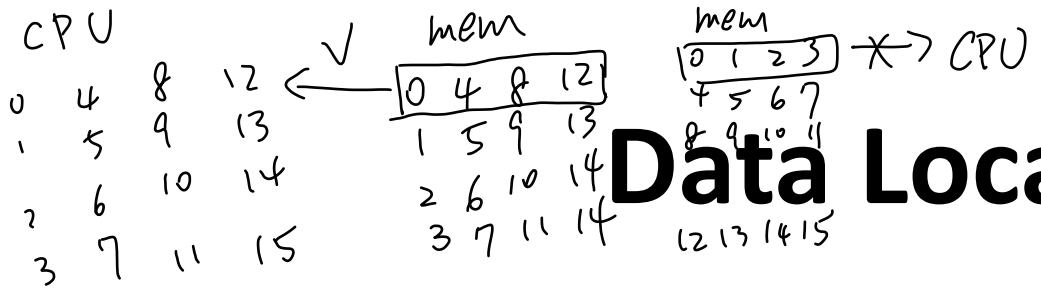
動態負載平衡

Dynamic Load Balancing

- Assigns work among processors at runtime
- Better for heterogeneous multicore
- Dynamic load balancing is needed when static load balancing is difficult, e.g., Sparse arrays

•在運行時在處理器之間分配工作
•更適合異構多核
•當難以進行靜態負載平衡（例如，稀疏陣列）時，需要動態負載平衡

如果 Data 少 ⇒ 用 single
可以連續的話盡量（陣列的部份）



Data Locality 數據局部性

- Principle of data locality:
  - Task accesses a relatively small portion of the address space at continuous time
- Temporal locality (locality in time)
  - e.g. instruction and data in a loop
  - Parallel computation is serialized due to memory contention and lack of bandwidth
- Spatial locality (locality in space)
  - e. g. instruction are normally accessed sequentially, good spatial locality
  - how to allocate tasks and assign data to cores

•數據局部性原則：
–任務在連續時間內訪問相對較小的地址空間

•時間地點（時間地點）
–例如 指令和數據循環
–由於內存爭用和缺乏帶寬，並行計算被串行化

•空間局部性（空間局部性）
–例如 指令通常按順序訪問，具有良好的空間局部性
–如何分配任務並將數據分配給核心

4/14

Performance of Parallel Computing

並行計算的性能
- •覆蓋範圍
- •粒度
- •同步
- •通訊
- • 負載均衡
- •地區

- Coverage
- Granularity
- Synchronization
- Communications
- Load balance
- Locality