CH8、Multiprocessor

多重處理器

重點一：多重處理器的基本概念

定義：至少含兩個以上處理器的單一電腦。可經由工作層級平行(Job-level parallelism)、或行程層級平行(process-level parallelism)，於獨立的工作間獲得最高的 throughput，並且可藉由多重處理器，執行平行處理程式(parallel processing program)，來改善單一程式的執行時間。

晶片多核心微處理機(Chip Multicore micro Processors, CMPs)：現今的電腦在單一積體電路中，通常都包含一個以上核心(core)。而處理器核心數，預期會每隔兩年、成長一倍

		SW	
		Sequential *(在 run time 不可被平行分解)*	Concurrent *(在 run time 可被平行分解)*
HW	Serial*(只有一個處理器)*	Matrix Multiply written in Matlab running on an Intel Pentium 4	Windows Vista Operating System running on an Intel Pentium 4
	Parallel*(有多個處理器可平行執行)*	Matrix Multiply written in Matlab running on an Intel Xeon e5345 (Clovertown)	Windows Vista Operating System running on an Intel Xeon e5345 (Clovertown)

Cluster 叢集：將獨立的工作站或個人電腦，透過區域網路連接，使其在功能上像是一個大型的多處理器，總用於：搜尋引擎、網頁伺服器、郵件伺服器、資料庫…等。特大型的 Cluster 稱為 Warehouse Scale Computer, WSC

重點二：撰寫平行程式的挑戰

撰寫平行程式的挑戰包含：排程(Scheduling)、負載平衡(Load Balance)、同步時間(Time of Synchronization)、溝通成本(Overhead of Communication)

應用上幾乎不可能完全平行化。假設 s 是一件工作必須循序處理的比例，因此(1-s)便是可平行處理的比例；假設 P=處理器個數

Speedup(P) = EXTime(1) / EXTime(P)

<= 1 / [s+(1-s/P)] <= 1/s

即使應用可平行部分加速地非常完美，其效能仍受限於循序部分的比例

練習：Suppose you want to achieve a speed-up of 90 times faster with 100 processors. What percentage of the original computation can be sequential?

90 = 1 / [(1-f) + f/100] => f=0.99；Thus, the sequential percentage can only be 0.1%

練習：Suppose you want to perform two sums: one is a sum of 10scalar variables, and one is a matrix sum of a pair of two-dimensional arrays, with dimensions 10 by 10. What speed-up do you get with 10 versus 100 processors? Next, calculate the speed-ups assuming the matrices grow to 100 by 100.

Assume that the time for an addition is t.
*10*10 matrix*
*Single processor: Execution Time = (9+100) * t = 109t*
*10 processors: Execution Time = 9*t + (100/10)*t = 19t*
Speedup = 109/19 = 5.74
*100 processors: Execution Time = 9*t + (100/10)*t = 10t*
Speedup = 109/10 = 10.9
*100*100 matrix*
*Single processor: Execution Time = (9+10000) * t = 10009t*
*10 processors: Execution Time = 9*t + (10000/10)*t = 1009t*
Speedup = 10009/1009 = 9.92
*100 processors: Execution Time = 9*t + (10000/100)*t = 109t*
Speedup = 10009/109 = 91.83

使用多重處理器來增加效能、但不改變原本問題的大小，是相對困難的，因此，此種加速方式可分成：

1. Strong Scaling：when speedup can be achieved on a multiprocessor without increasing the size of the problem
2. Weak Scaling：when speedup is achieved on a multiprocessor by increasing the size of the problem proportionally(等比例) tot the increase in the number of processors.

練習：To achieve the speed-up of 91 on the previous larger problem with 100 processors, we assumed the load was perfectly balanced. That is , each of the 100 processors had 1% of the work to do. Instead, show the impact on speed-up if one processor’s load is higher than all the rest. Calculate at 2% and 5%.

*If one processor has 2% of the parallel load, then it must do 2%*10000 or 200 additions, and the other 99 will share the remaining 9800.*
ExTime(1) = 9t + 10000t = 10009t
ExTime(100) = 9t + MAX(9800t/99 , 200t/1) = 9t+200t = 209t
Speedup(100) = 10009t / 209t = 47.89

負載平衡(Load Balancing)是另一項重要因素。以上範例可知，只要其中一處理器的負載為其他處理器的兩倍，則整個系統能獲得的加速幾乎是減半的

練習：True or False：To benefit from a multiprocessor, and application must be concurrent.

Job-level parallelism can help sequential applications and sequential applications can be made to run on parallel hardware, although it is more challenging.

例(8)：Please answer True or False to each of the following statements.
1. More powerful instructions mean higher performance.
2. Computers at low utilization imply low power consumption.
3. To benefit from a multiprocessor, an application must be concurrent.
4. GPUs reply on graphics DRAM chips to reduce memory latency and thereby increase performance on graphics applications.
5. There is no way to reduce compulsory misses.

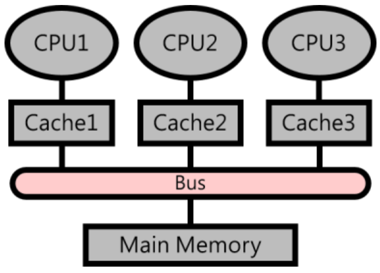
1. *False*
2. *False (low frequency imply lower power consumption)*
3. *False*
4. *False (increase bandwidth not reduce latency)*
5. *False*

練習：True or False：Strong scaling is not bound by Amdahl's law.

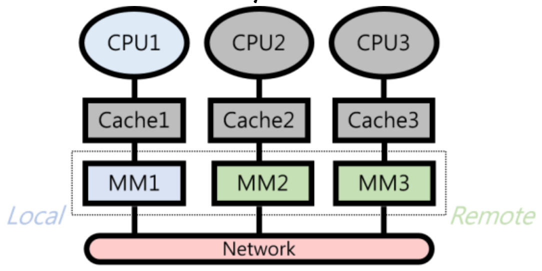
False. Weak scaling can compensate for a serial portion of the program that could otherwise limit scalability.

重點三：記憶體共享之多處理器(SMP)
定義：多處理器可把任務分成多個平行執行緒，需共享記憶體資料

分成：
1. UMA, Uniform Memory Access multiprocessors(又稱 Symmetric multiprocessor, SMP)：所有處理器存取 Memory 的時間一致



2. NUMA, Non-uniform Memory Access multiprocessor：所有處理器存取 Memory 的時間不一致*(Local 較快、Remote 較慢)*，規模可較大，但設計較複雜：
   1. Synchronization (Data Race, Critical Section)
   2. Cache Coherency

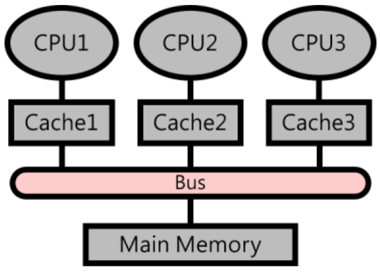


例(1)：Please explain the following terms in English. For each term, please spend no more than 100 English words.
1. Non-uniform memory access architecture
2. GPGPU

1. *Non-uniform memory access architecture: A type of single address space multiprocessor in which some memory accesses are faster than others depending on which processor asks for which word.*
2. *GPGPU: Using a GPU for general-purpose computation via a traditional graphics API and graphics pipeline.*

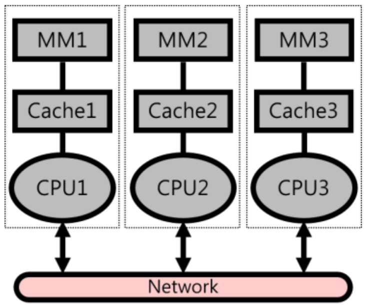
例(20)：In parallel processors sharing data, answer the following:
1. In uniform memory access (UMA) designs, do all processors use the same address space?
2. Do all UMA processors access memory at the same speed?
3. Draw a system diagram showing how processors and memory (modules) are connected.

1. *Yes*
2. *Yes*
3.



重點四：傳遞訊息之多處理器(MPP)

定義：每一個處理器有自己一組 Cache 與 Memory，資料的共享，是以靠明確的 Send()與 Receive()的方式，建立通道來傳遞，故速度較慢，但硬體較簡單



例(23)：
1. What are the two possible approaches for parallel processors to share data?
2. Outline Flynn’s taxonomy of parallel computers.
3. Suppose you want to perform two sums: one is a sum of two scalar variables and one is a matrix sum of a pair of two-dimensional arrays, size 500 by 500. What speedup do you get with 500 processors?

1. Shared-memory and Message passing
2. (1)Single instruction stream, single data stream (SISD, the uniprocessor)
(2)Single instruction stream, multiple data streams (SIMD)
(3)Multiple instruction streams, single data stream (MISD)
(4)Multiple instruction streams, multiple data streams (MIMD)
*3. Speedup = (1 + 500*500) / (1 + 500*500/500) = 499*

例(22)：Which of the following architecture/model for multiprocessors systems is most unlikely to adopt techniques of critical sections that are commonly discussed in the course of Operating System? Explain your answer.
1. Uniform memory access
2. Symmetric multiprocessors
3. Non-uniform memory access
4. Message passing

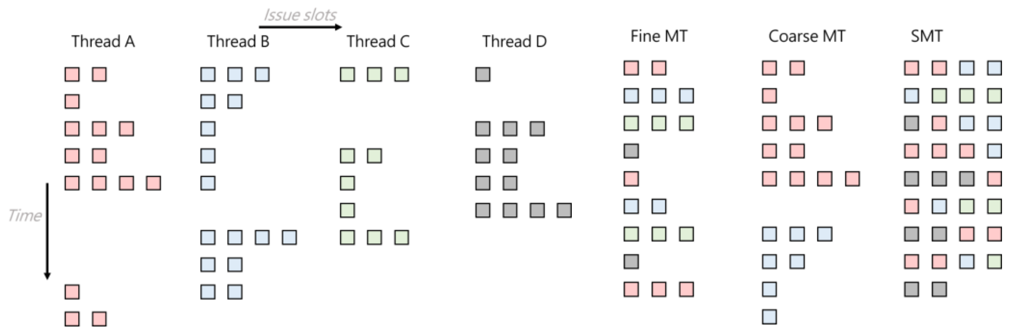
4
註：1、2、3 all belong to single-bus multiprocessor system and one of the major requirements of a single-bus multiprocessor is to be able to coordinate processes that are working on a common task. That is synchronization. Critical section is a technique which deals with synchronization problem.

重點六：單晶片之多重執行緒

(不一定是要多處理器，單處理器亦有)

利用多執行緒的切換，讓處理器使用率提高，主要有 2 種方式，差別在切換 Thread 的時機：

1. Fine-grained Multithreading：輪流讓多個執行緒執行
   (1) 優點：可隱藏較長與較短 stall 的 throughput losses
   (2) 缺點：會使單一執行緒慢下來
2. Coarse-grained Multithreading：
   (1) 優點：較不會使單一執行緒慢下來
   (2) 缺點：會有 pipeline start-up 成本，因為在 Thread 交換的時候，會需要清掉與重新填入 thread
3. Simultaneous Multithreading(SMT)：屬於硬體多執行緒的一種變形，利用到兩種技術：動態排程處理器(dynamically scheduler process, superscalar)、與程式的指令層級平行(ILP, Instruction Level Parallel)和執行緒層級平行(TLP, Thread Level Parallel)



例(12)：Please explain the following terms:
1. Simultaneous multithreading
2. Thread-Level Parallelism
3. Branch target buffer (BTB)
4. Precise interrupt
5. Branch prediction

1. *Simultaneous multithreading:*
   A variation on hardware multithreading that uses the resources of a multiple-issue, dynamically scheduled processor (superscalar) to exploit both program ILP and thread-level parallelism (TLP).
2. *Thread-Level Parallelism:*
   Multiple thread contexts in CPU.
3. *Branch target buffer (BTB):*
   A structure that caches the destination PC or destination instruction for a branch. It is usually organized as a cache with tags, making it more costly than a simple prediction buffer.
4. *Precise interrupt:*
   An interrupt that is always associated with the correct instruction in pipeline computers.
5. *Branch prediction*
   Prediction of branches at runtime

例(40)：
1. Some multicore designers claim their chips can attain a linear speedup with processor numbers. Please describe the possible problem set that fit such claim.
2. True or False, RAID systems rely on redundancy to achieve high reliability.
3. True or False, in the simultaneous multithreading case, both thread-level parallelism and data parallelism are exploited for parallel execution.
4. Given the following instruction sequence of three threads, (the vertical axis is time, the horizontal axis is issue slot) how many clock cycles will fine-grained multithreading and simultaneous multithreading use respectively? Annotation [] denotes the issue slot. Assume maximum four issue slots are allowed per cycle.

Thread 1	Thread 2	Thread 3
[]	[][][]	[][]
[][]	[][][][]	[][][]
[][][]	[][]	(stall)
[][]	(stall)	(stall)
(stall)	[]	[][][][]
[]	[][][]	[]
[][][][]	(stall)	[]
[][][]	[][]	[][]
[]	(stall)	[]
	[][][]	

1. *The problem set that fit such claim should possess the following features:*
   -100% Parallelizable
   -No data dependency between processes (threads)
2. *False. (Availability)*
3. *False. (Thread-level parallelism and Instruction parallelism)*
4. *Fine=grained multithreading: 22 clocks*
   Simultaneous multithreading: 13 clocks ([49/4] = 13)

練習：True or False
1. Both multithreading and multicore rely on parallelism to get more efficiency from a chip.
2. Simultaneous multithreading uses threads to improve resource utilization of a dynamically scheduled, out-or-order processor.

1.True、2.True

練習：What are the advantages & disadvantages of fine-grained multithreading, coarse-grained multithreading, and simultaneous multithreading?

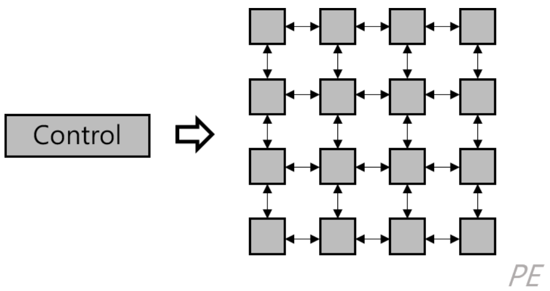
1. *Fine-grained multithreading:*
   Can hide the throughput losses arises from both short and long stalls. But it will slow down the execution of individual threads, especially those without stalls.
2. *Coarse-grained multithreading:*
   Only switches those when there is a long stall, so it is less likely to slow down the execution of individual threads. However, it has limited ability to overcome throughput losses due to short stalls and relatively higher startup overhead.
3. *Simultaneous multithreading:*
   Dynamically issue operations from multiple threads simultaneously. Tis covers the throughput losses from both short and long stalls, and does not suffer from high switching overhead. But SMT may still slow down the execution of individual threads if that thread does not have any stall.

重點七：平行電腦的分類

(由 Flynn 分類)

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD： Intel Pentium 4	SIMD： SSE instructions of x86
	Multiple	MISD： No examples today	MIMD： Intel Xeon e5345

SISD：傳統電腦(uniprocessor)
SIMD：多個處理器受一個控制器管理(可同時處理不同資料)



MISD：現今無實作出之電腦
MIMD：一般的平行處理
SPMD(Program)：一個 program 複製到各個處理器平行執行

Vector Processor(向量處理器)：
與 SIMD 類似，皆擅長執行平行處理(ex：命為 Cray 的超級電腦)

例(13)：Compare the properties and architectures between SISD, SIMD, MISD and MIMD

SISD (Single Instruction Single Data):

Machines are conventional serial computers that process only one stream of instructions and one stream of data.

SIMD (Single Instruction Multiple Data):

Machines which have many identical interconnected processors under the supervision of a single control unit. All the processing elements simultaneously execute the same instruction. Each processor works on data from its own memory and hence on distinct data streams.

MISD (Multiple Instruction Single Data):

Machines have many processing elements, all of which execute independent streams of instructions. However, all the processing elements work on the same data stream.

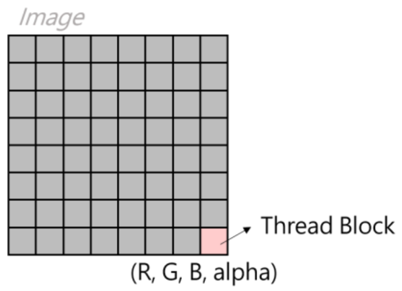
MIMD (Multiple Instruction Multiple Data):

Machine has many interconnected processing elements, each of which has their own control unit. The processors work on their own data with their own instructions.

重點八：圖形處理器

2008 年，因經融海嘯大裁員，許多人失業在家只能打電動，卻意外造就了圖形處理器 GPU 的大幅發展

定義：非單一處理器，而是大數量的處理器組合而成，愈高級的 GPU 數量愈多，可使用 Hardware Multithreading 方式，加速運算。*(多使用 SIMD processor)*



GPU 相對於 CPU 之特性：

1. GPU 只是 CPU 的加速輔助
2. 程式語言：使用 High-level API，例如：OpenGL, Cg, HLSL…等
3. 只具有 2 種 Data Type：1.Vertices(x, y, z, w)、2.Pixels (R, G, B, alpha)
4. 使用元素圖：3D 幾何圖型，例如：線、三角型、陰影…等
5. Working set：上百個 Mega-bytes

GPU 與 CPU 架構的不同點：

1. CPU 注重 Multi-level Cache 來隱藏記憶體延遲，但 GPU 注意足夠的 Threads 來隱藏記憶體延遲
2. 相對於『延遲』，GPU 記憶體更重視『頻寬』
3. 早期的 GPU 有較單一功能，效能較差；現今 GPU 更傾向具有一般功能的處理器、與 Scalar Instruction，以增加其可程式性
4. GPU 沒有倍精度浮點數

General Purpose GPU：

通常 GPU 都是設計來執行特定功能的運算，但有些工程師希望它們能有更好的效能與更多元的功能，因此為了區分此種更具一般性功能的 GPU，即稱之為 GPGPU

Nvidia 的 CUDA(Compute Unified Device Architecture)即可以使用 C 語言，就能編寫 GPU 的平行程式

例(10)：The flowing acronyms are commonly used in the computer architecture literature. Please briefly explain their meanings.

1. AMAT
2. VLIW
3. RISC
4. SIMD
5. GPGPU

1. *AMAT (average memory access time) is the average time to access memory considering both hits and misses and the frequency of different accesses.*
2. *VLIW (very long instruction word)：A style of instruction set architecture that launches many operations that are defined to be independent in a single wide instruction, typically with many separately opcode fields.*
3. *RISC (Reduced instruction set computer) is a computer which is based on the design strategy that simplified instructions can provide higher performance if this simplicity enables much faster execution of each instruction.*
4. *SIMD (Single Instruction stream, Multiple Data streams)：A multiprocessor. The same instruction is applied to many data streams, as in a vector processor or array processor.*
5. *GPGPU (General-purpose GPU)：Using a GPU for general-purpose computation via a traditional graphics API and graphics pipeline.*

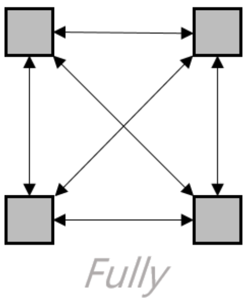
例(8)：Please answer True of False to each of the following statements.
1. More powerful instructions mean higher performance.
2. Computers at low utilization imply low power consumption.
3. To benefit from a multiprocessor, an application must be concurrent.
4. GPUs reply on graphics DRAM chips to reduce memory latency and thereby increase performance on graphics applications.
5. There is no way to reduce compulsory misses.

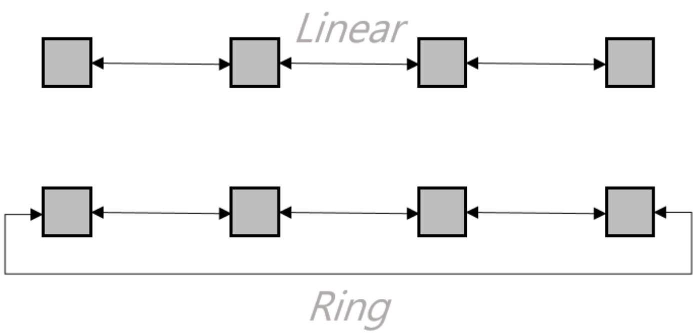
1. *False*
2. *False (low frequency imply lower power consumption)*
3. *False*
4. *False (increase bandwidth not reduce latency)*
5. *False*

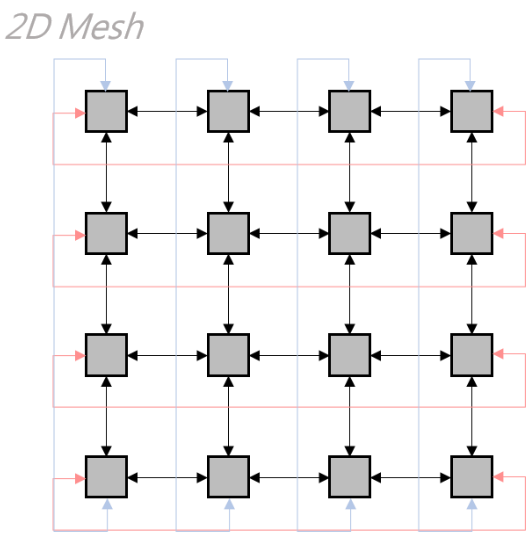
重點九：網路拓撲學

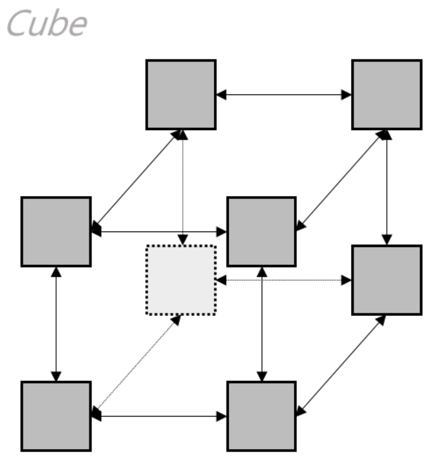
Network Topology						
Single Stage					Multiple Stage	
Linear	Ring	Fully	2D Mesh	N-Cube	Crossbar	Omega Network

Single Stage：









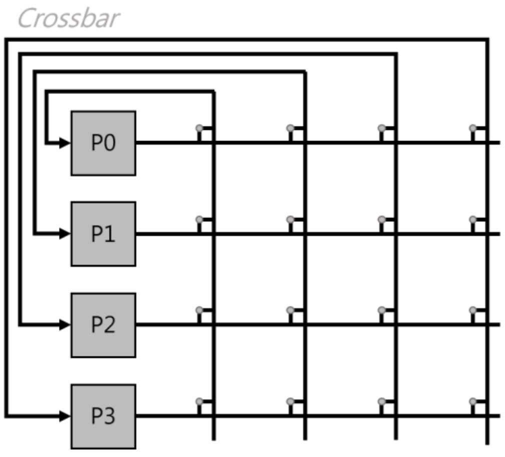
1. Diameter：兩點之間的最長最短路徑(2-cube=3)[效能]
2. Nodal degree：最多一點的接點數(2-cube=3)[容錯]
3. Network bandwidth：總傳輸資料量(2-cube=12B)[效能]
4. Bisection bandwidth：切成 2 半後，少掉的傳輸資料量(2-cube=8B)[容錯]

練習：Besides network bandwidth and bisection bandwidth, two other properties sometimes used to describe network typologies are the diameter and the nodal degree. The diameter of a network is defined as the longest minimal path possible, examining all pairs of nodes. The nodal degree is the number of links connecting to each node. If the bandwidth of each link in a network is B, find the diameter, nodal degree, network bandwidth, and bisection bandwidth for the 2D grid and n-cube tree.

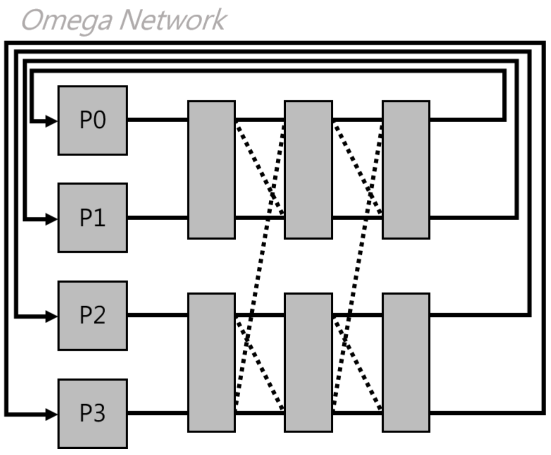
1. *2D grid/mesh:*
   Diameter: 4
   Nodal degree: 4
   *Network Bandwidth: 2*P*B = 2*16*B = 32B*
   Bisection Bandwidth: 8B
2. *N-cube tree(n=3):*
   Diameter: 3
   Nodal degree: 3
   Network Bandwidth: 12B
   Bisection Bandwidth: 4B

Multiple Stage：

1. Crossbar：利用開關通道來選擇

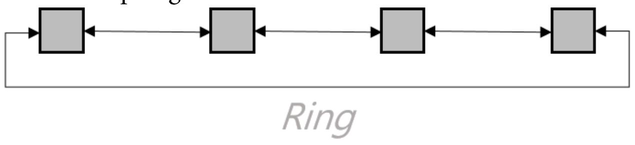


2. Omega network：利用 Cross 與 Bar 來選擇



例(14)：Costs of network topologies include the number of switches, the number of links on a switch to connect to the network, the data width per link, etc. Its performance includes the latency on an unloaded network to send/receive a message, the throughput, delays caused by contention, etc. Networks are normally drawn as graphs, with each arc representing a link. The processor-memory node is shown as a black square, and the switch is shown as a circle.

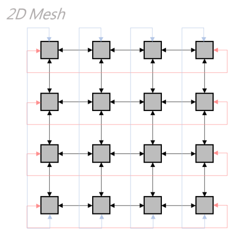
1. For single-stage networks, one processor is placed at every switch node. For example, one of popular network topologies is ring (see the following figure). List and draw two other single-stage network topologies.

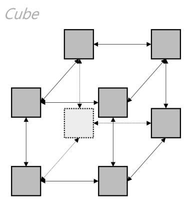


2. On the other hand, multistage networks include Crossbar and Omega network, where processors and switches are separate. Assume the switch node controls whether the unidirectional links, A and B, connect to each other or not, as shown in the following figure.

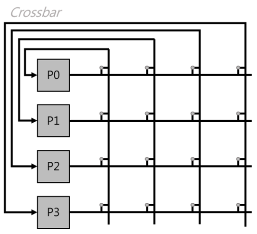
Consider there are 8 processors, draw the Crossbar and Omega network topologies, list the number of switches required, and compare the advantage, disadvantage between the two topologies.

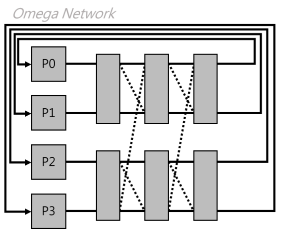
1. 2D、n-cube





2. Crossbar、Omega network

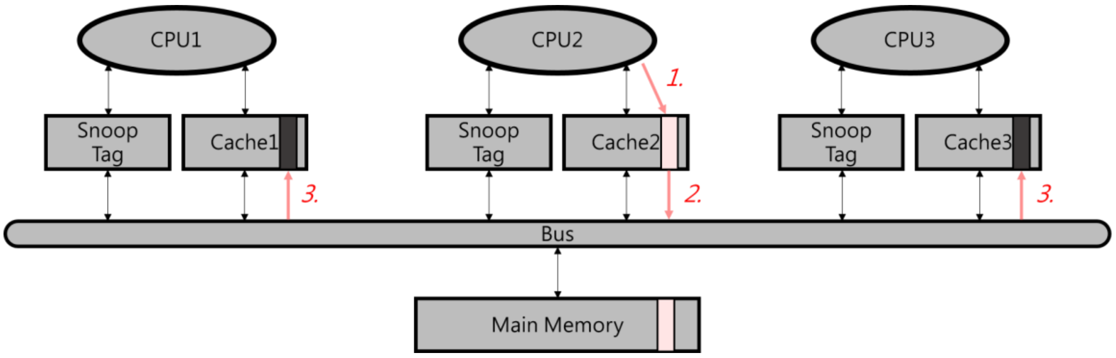




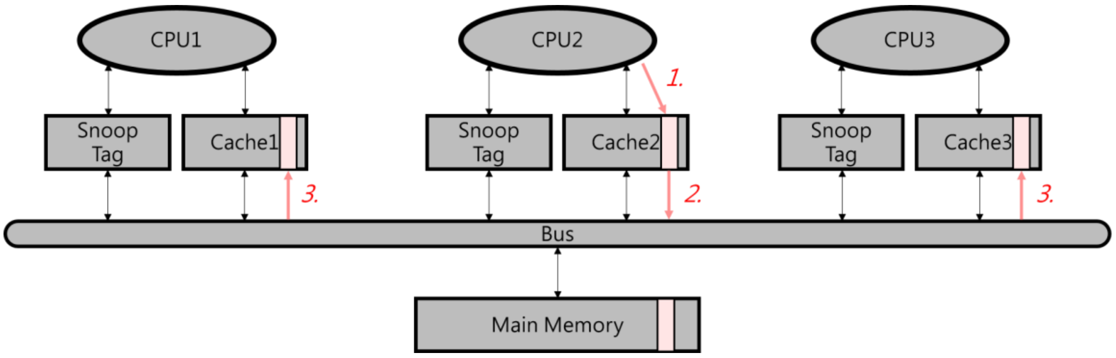
重點五：多處理器的快取一致性

UMA：snooping(最受歡迎) *//NUMA 使用 Directory*

1. Write invalidate：1.CPU 寫入、2.透過 Bus、3.將其他 Cache 資料、設定為無效(Invalid)。需要 Snoop Tag 協助；商用機器常用*(常搭配 Write Back 至記憶體)*



2. Write update (Write broadcast)：1.CPU 寫入、2.透過 Bus、3.寫至其他 Cache、更新資料。需要 Snoop Tag 協助



例(18)：”Snoopy” protocol is a mechanism for multiprocessors system. Please specify why or where snoopy protocol is used.

Multiple processors require copies of the same data in multiple caches. This will result in the problem in consistency of data between the versions in the caches of several processors. The protocols to maintain coherence for multiple processors are called cache coherence protocols. Snoopy protocol is used to maintain cache coherence in multiprocessors system.

例(21)：(前略)
1. Briefly describe each type of the snooping protocol.
2. Which type has less demand on bus bandwidth? Explain your answer

1. *Write invalidate: The writing processor causes all copies in other caches to be invalidated before changing its local copy; the writing processor issues an invalidation signal over the bus, and all caches check to see if they have a copy; if so, they must invalidate the block containing the word.*

   Write update: Rather than invalidate every block that is shared, the writing processor broadcasts the new data over the bus; all copies are then updated with the new value. This scheme, also called write broadcast, continuously broadcasts writes to shared data, while write invalidate deletes all other copies so that there is only one local copy for subsequent writes.

2. *Write invalidate has less demand on bus bandwidth. Because write update forces each access to shared data always to go around the cache to memory, it would require too much bus bandwidth.*

例(2)：Which of the followings are true about multiprocessors?
1. Communications among different processors cannot be achieved by centralized shared memory.
2. Distributed shared-memory scheme might result in non-uniform memory access time.
3. Multiple instruction stream multiple data stream scheme offers flexible and cost performance advantages.
4. Memory Coherency can be achieved by snooping protocols.

2、3、4
註(2)：NUMA architecture also called Distributed Shared Memory.

例(7)：Please explain the following terms:
1. Parallel processing program
2. Process-level parallelism
3. Instruction-level parallelism
4. Cache coherence

1. *Parallel processing program: A single program that runs in multiple processors simultaneously.*
2. *Process-level parallelism: Utilizing multiple processors by running independent programs simultaneously*
3. *Instruction-level parallelism: The parallelism among instructions*
4. *Cache coherence: Consistency in the value of data between the versions in the caches of several processors.*

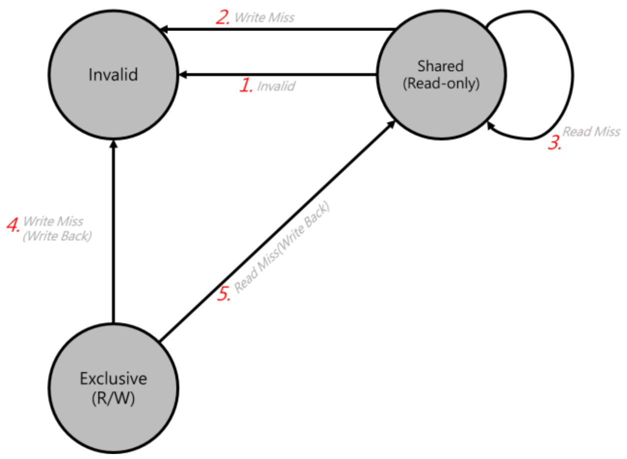
快取一致性協定的範例

四個假設前提：
1. UMA
2. Write-Invalidate
3. Write Back
4. Direct-Mapping

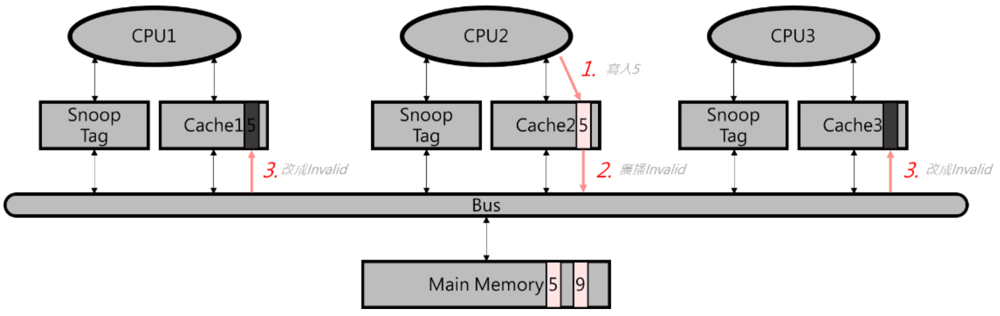
Block 的三種狀態：

1. 共享 Shared：表示此快取是乾淨的，區塊中的資料沒被修改過、可共享
2. 互斥 Exclusive：表示此快取是髒的(Dirty)，資料被修改過、不可共享
3. 無效 Invalid：表示此區塊資料為無效

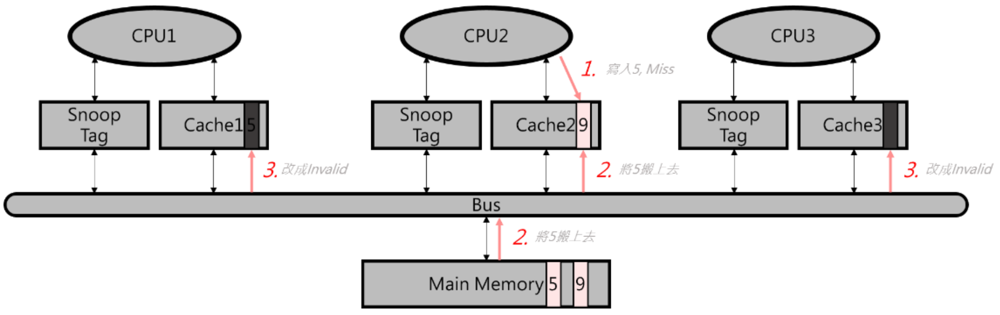
Cache State transitions using signals from the bus



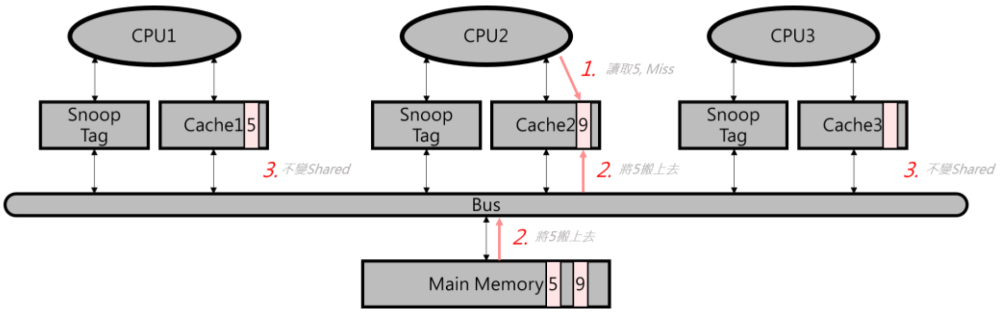
1. Shared -> Invalid



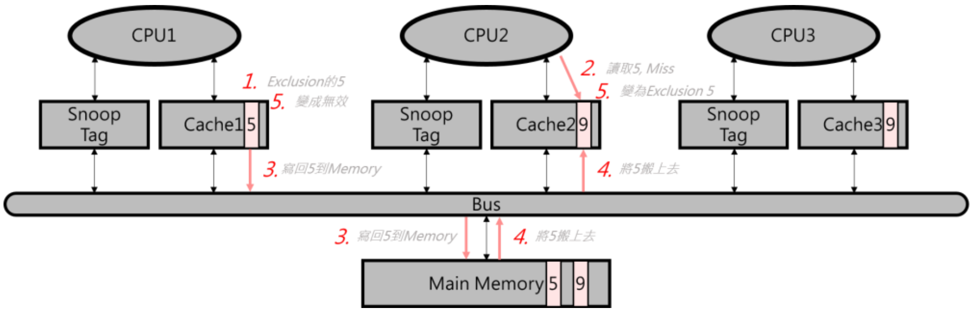
2. Shared -> Invalid



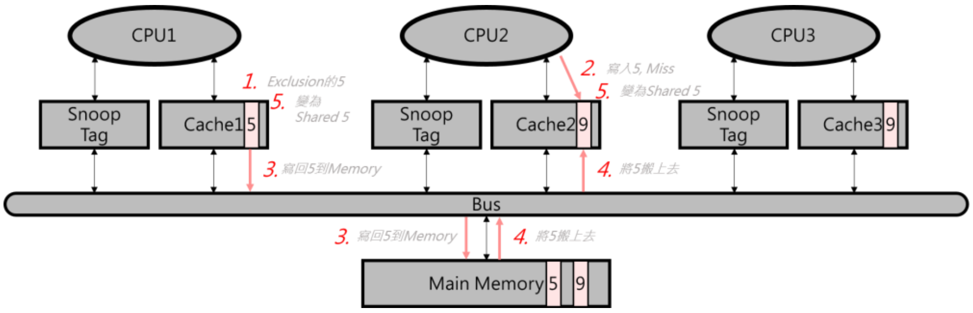
3. Shared -> Shared



4. Exclusive -> Invalid

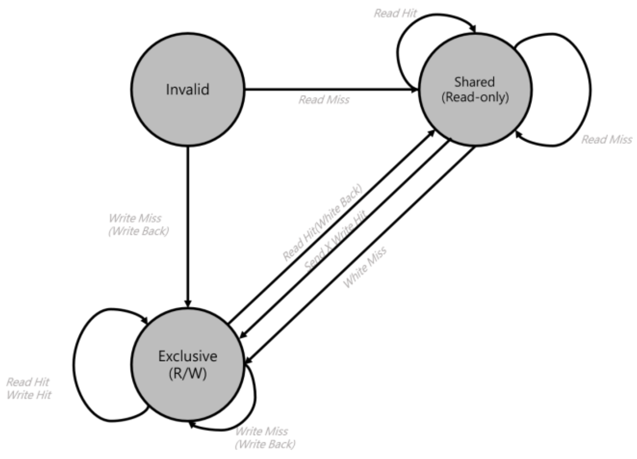


5. Exclusive -> Shared



Cache State transitions using signals from the Processor

(與上雷同，但只單看 Processor 部分)



例(6)：You are given a bus-based multiprocessor system which implements a snooping-based cache coherency protocol. There are two processors P1, P2 in the system. Each processor has its own L1 cache. The memory system specification is given below:

-L1 Cache: physical addressed, 64KB, direct-mapped, 32B block size, write-back cache

-Virtual Memory: 8KB page, 1GB virtual address space

-TLB: fully associative, 128 entries

-64MB of addressable physical memory.

x1, x2, and x3 represent one-word variables. The physical addresses (represented in hex) of x1, x2, and x3 are 0x00AF0804, 0x00AF0808, and 0x00AE0804, respectively. Answer the following questions considering the possible events: TLB-hit/miss, page-table hit/miss, L1-hit/miss, write-back invalidate.

1. What are the sizes of the tag and data arrays of the TLB?
2. Please write down the correct order of the events listed above for detecting a page fault. Not that some events may not occur.
3. Assume TLB hits for all the memory requests. x1 and x2 are in the L1 caches of P1 and P2. Show the bus activities for each memory request.

Time	P1	P2	Bus activity
1	Write*1		
2		Read*2	
3	Write*1		
4		Read*3	
5	Read*2		

1. *The size of a Tag = 30-13 = 17 bits*
2. *TLB miss, page table miss*
3. *Variable x1 and x2 is contained in the same block since block size Is 32B.*

   Write-invalid snooping protocol:

Time	*P1*	*P2*	*Bus activity*
1	*Write*1*		*Yes*
2		*Read*2*	*Yes*
3	*Write*1*		*Yes*
4		*Read*3*	*Yes*
5	*Read*2*		*No*

註：*Write-update snooping protocol:*

Time	*P1*	*P2*	*Bus activity*
1	*Write*1*		*Yes*
2		*Read*2*	*No*
3	*Write*1*		*Yes*
4		*Read*3*	*Yes*
5	*Read*2*		*No*

例(9)：Consider a bus-based shared memory system consisting of three processors. The shared memory is divided into four blocks, a, b, c, d. Each processor has a cache that can fit only one block at any given time. Each block can be in one of two states: valid(V) or invalid(I). Assume that caches are initially flushed (empty) and that the contents of the memory are as follows:

Memory block	Contents
a	10
b	20
c	40
d	80

Consider the following sequence of events given in order:

1.	P1: read(a)	3.	P3: read(a)	5.	P1: read(c)	7.	P3: a=15
2.	P2: read(a)	4.	P1: a=a+20	6.	P2: read(a)	8.	P1: c=c+10

1. There are two fundamental cache coherence policies:
   Write-invalidate and write-update. Write-invalidate maintains consistency by reading from local caches until a write occurs. When any processor updates the value of X through a write, it invalidates all other copies by setting the state of the block to invalid(I). Suppose write-back and write-invalidate protocols are used in the system, what would be the contents of the caches and memory and the state of cache blocks after the above sequence?
2. On the other hand, write-update maintains consistency by immediately updating all copies in all caches. When a block is updated, the state of the block is set to valid(V). Suppose write-through and write-update protocols are used in the system, what would be the contents of the caches and memory and the state of cache blocks after the above sequence?

1.

Event	*Cache 1*		*Cache 2*		*Cache 3*		*Memory block*			
	state	*value*	*state*	*value*	*state*	*value*	*a*	*b*	*c*	*d*
Initial	*I*		*I*		*I*		*10*	*20*	*40*	*80*
1. P1: read(a)	*V*	*10*	*I*		*I*		*10*	*20*	*40*	*80*
2. P2: read(a)	*V*	*10*	*V*	*10*	*I*		*10*	*20*	*40*	*80*
3. P3: read(a)	*V*	*10*	*V*	*10*	*V*	*10*	*10*	*20*	*40*	*80*
4. P1: a=a+20	*V*	*30*	*I*		*I*		*10*	*20*	*40*	*80*
5. P1: read(c)	*V*	*40*	*I*		*I*		*30*	*20*	*40*	*80*
6. P2: read(a)	*V*	*40*	*V*	*30*	*I*		*30*	*20*	*40*	*80*
7. P3: a=15	*V*	*40*	*I*		*V*	*15*	*30*	*20*	*40*	*80*
8. P1: c=c+10	*V*	*50*	*I*		*V*	*15*	*30*	*20*	*40*	*80*

2. 表

Event	*Cache 1*		*Cache 2*		*Cache 3*		*Memory block*			
	state	*value*	*state*	*value*	*state*	*value*	*a*	*b*	*c*	*d*
Initial	*I*		*I*		*I*		*10*	*20*	*40*	*80*
1. P1: read(a)	*V*	*10*	*I*		*I*		*10*	*20*	*40*	*80*
2. P2: read(a)	*V*	*10*	*V*	*10*	*I*		*10*	*20*	*40*	*80*
3. P3: read(a)	*V*	*10*	*V*	*10*	*V*	*10*	*10*	*20*	*40*	*80*
4. P1: a=a+20	*V*	*30*	*V*	*30*	*V*	*30*	*30*	*20*	*40*	*80*
5. P1: read(c)	*V*	*40*	*V*	*30*	*V*	*30*	*30*	*20*	*40*	*80*
6. P2: read(a)	*V*	*40*	*V*	*30*	*V*	*30*	*30*	*20*	*40*	*80*
7. P3: a=15	*V*	*40*	*V*	*15*	*V*	*15*	*15*	*20*	*40*	*80*
8. P1: c=c+10	*V*	*50*	*V*	*15*	*V*	*15*	*15*	*20*	*50*	*80*

例(24)：Multiprocessor designs have become popular for today's desktop and mobile computing. Given a 2-way symmetric multiprocessor (SMP) system where both processors use write-back caches, write update cache coherency, and a block size of one 32-bit word. Let us examine the cache coherence traffic with the following sequence of activities involving shared data. Assume that all the words already exist in both caches and are clean. Fill-in the last column (a)-(e) in the table to identify the coherence transactions that should occur on the bus for the sequence.

Step	Processor	Memory activity	Memory address	Transaction required (Yes or No)
1	Processor 1	1-word write	100	(a)
2	Processor 2	1-word write	104	(b)
3	Processor 1	1-word read	100	(c)
4	Processor 2	1-word read	104	(d)
5	Processor 1	1-word read	104	(e)

(a)	*(b)*	*(c)*	*(d)*	*(e)*
Y	*Y*	*N*	*N*	*N*

例(25)：False sharing can lead to unnecessary bus traffic and delays. Follow the direction of above question, except change the cache coherency policy to write-invalidate and block size to four words (128-bit). Reveal the coherence transactions on the bus by filling-in the last column (a)~(e) in the table below.

Step	Processor	Memory activity	Memory address	Transaction required (Yes or No)
1	Processor 1	1-word write	100	(a)
2	Processor 2	1-word write	104	(b)
3	Processor 1	1-word read	100	(c)
4	Processor 2	1-word read	104	(d)
5	Processor 1	1-word read	104	(e)

(a)	*(b)*	*(c)*	*(d)*	*(e)*
Y	*Y*	*Y*	*N*	*N*

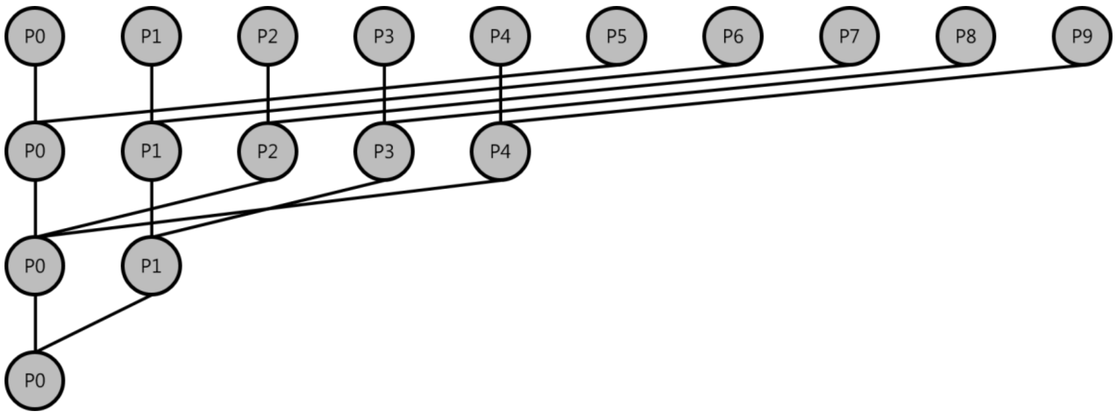
練習(310)：Sum 100000 numbers on 100 processor UMA.
-Each processor has ID: 1<= Pn <= 99
-Partition 1000 numbers per processor
Initial summation on each processor

The first step would be to split the set of numbers into subsets of the same size.

```
Sum[Pn] = 0;
for(i=1000*Pn; i<1000*(Pn+1); i=i+1)
    sum[Pn]=sum[Pn]+A[i];
```

The next step is to add these many partial sums

```
half=100;
repeat
    synch();
    if(half%2!=0 && Pn==0)
        sum[0]=sum[0]+sum[half-1];
    half=half/2;
    if(Pn<half)
        sum[Pn]=sum[Pn]+sum[Pn+half];
until (half==1);                    //final sum in sum[0]
```



例(3)：Suppose we want to sum 10000 numbers on a single-bus multiprocessor computer. Let's assume we have 10 processors. The first step would be to split the set of numbers into subsets of the same size. All processors start the program by running the following loop that sums their subset of numbers, where Pn is its processor index(0~9):

```
Sum[Pn]=0;
for(i=1000*Pn; i<1000*(Pn+1); i=i+1)
    sum[Pn]=sum[Pn]+A[i];
```

The next step is to add these many partial sums, so we divide to conquer. Half of processors add pairs of partial sums, then a quarter add pairs of the new partial sums, and so on until we have the single, final sum. We want each processor to have its own version of loop counter variable I, so we must indicate that it is a "private" variable. In this problem, the two processors must synchronize before the "consumer" processor tries to read the result from the memory location written by the "producer" processor; otherwise, the consumer may read the old value of the data. Here is the code (half is also a private varable):

```
half=10;
repeat
    synch();
    if(half%2!=0 && Pn==0)
        sum[0]=sum[0]+sum[half-1];
    half=half/2;
    if(Pn<half)
        sum[Pn]=sum[Pn]+sum[Pn+half];
until (half==1);
```

Question: according to the algorithm, find out what operations are executed by the designated processor during the designated repeat-loop iteration. (Ex: NOP or sum[0]=sum[0]+sum[4]). Copy Table 1 to your answer sheet and fill in the answers.

Process index	Repeat-loop iteration	Operation
Pn=2	First	
Pn=2	Second	
Pn=0	Third	

Process index	*Repeat-loop iteration*	*Operation*
Pn=2	*First*	*sum[2]=sum[2]+sum[7]*
Pn=2	*Second*	*NOP*
Pn=0	*Third*	*sum[0]=sum[0]+sum[1]*

例(11)：Parallel processing programming

Suppose that we want to sum 1000 numbers on a shared memory multiprocessor computer with uniform memory access time. Assume that we have 10 processors.

1. Assume we split the set of numbers into subsets of the same size for the processors to compute partial sums. Write a pseudo-code that computes the partial sums.
2. Write a pseudo-code that adds the partial sums.

1.

```
Sum[Pn]=0;
for(i=100*Pn; i<100*(Pn+1); i=i+1)
    sum[Pn]=sum[Pn]+A[i];
```

2.

```
half=10;
repeat
    synch();
    if(half%2!=0 && Pn==0)
        sum[0]=sum[0]+sum[half-1];
    half=half/2;
    if(Pn<half)
        sum[Pn]=sum[Pn]+sum[Pn+half];
until (half==1);
```