JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Computer Science 605.411

Module 12 Example Set 3

1. A certain program consists of a purely sequential part (Part 1) that must be executed before the remaining Part 2. Part 1 accounts for 24% of the instructions in the program. Part 2 can be split into four sections that correspond to 18%, 32%, 14% and 12% of the instructions in the program. The sections in part 2 are independent and can be executed in parallel. If each instruction in the program takes 1 cycle, what speedup would be provided by executing the program on a quad-core system versus a single core system? Round your answer to two decimal places.

Part 1 would take 0.24T time units. The longest section in part 2 takes 0.32T.
So the speedup = 1/(0.24+0.32) = 1/0.56 = 1.79

2. A program is to perform two sums: one is a sequential sum of 100 scalar variables and the other is the matrix sum of two 1000 by 1000 element arrays. Performing a single addition (the sum of two individual numbers) takes 1 cycle. The scalar sum must be computer before the matrix sum. What speedup is provided for the same program by a system that has 1000 processors compared to using a single processor?

Using one processor takes 99 + 1000000 = 1000099 cycles.
The sequential sum is computed in 99 cycles by a single processor. Each scalar variable has a different address that can not be computed based on a processor number.

The matrix sum would generate a matrix result such that CU,k] = AU,k] + BU,k].
Each matrix contains 1000000 elements. Each of the 1000 processors could compute the sum of the corresponding rows in the two matrices in parallel. This would take 1000 cycles to generate the matrix sum. So the total time would be 99 + 1000 = 1099.
The speedup = 1000099/1099 = 910.01

JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

3.A Quad-core system is used the compute the cummulative sum of the elements in an 8- element vector. Assume that the time required on any core to perform an addition is 2 nano- seconds. What speedup factor would this system provide compared to a single processor system if only the time required to perform the required additions is taken into account and no other overhead?

The cummulative sum = A[O] + A[l] + A[2] +A[3] + A[4] + A[S] + A[6] **+** A[7]

Partitioning the 8 elements into two groups of 4 elements each, a separate core could add a pair of elements in 2 ns to generate 4 partial sums:

A[O] + A[l], A[2] +A[3], A[4] + A[S], A[6] + A[7]

Then two cores could each add together 2 of these partial sums in 2 ns to generate 2 partial sums:

A[O] + A[l] + A[2] +A[3], A[4] + A[S] + A[6] + A[7]

Then one core would add the final 2 partial sums in 2 ns to generate the result. Thus a total 6 ns would be required.

A single core would have to perform 7 additions,which takes 14 ns. So the speedup = 14/6 = 2.33

4.A program that runs on one of the 3.3GHz processors within a NUMA multi-processor has an average CPI of 2. If an instruction references the remote memory, it takes an additional 300 ns. The program completes in 860 million cycles if none of the instructions reference the remote memory. How many additional cycles would the program take if 5% of the instructions reference the remote memory?

Since the average CPI=2, the instruction count= 860 million cycles / 2 cycles per instruction = 430 million instructions.

The 3.3 GHz clock rate corresponds to a 0.3 ns cycle time. Hence, 300ns corresponds to 300/0.3 = 1000 clock cycles. Therefore the number of extra cycles required if 5% of the instructions

reference the remote memory is 0.05* 430 million*lOOO = 21.5 billion cycles

5.Consider a multi-core processor with heterogenous cores: A, B C and D where core B runs twice as fast as A, core C runs three times as fast as A and core D runs at the same speed as core A.An application needs to execute a function that takes a single argument. Hence the function will be called once for each of the elements in an array of 256 elements. Core A requires T time units to execute the function for each array element.

a) Ifthe work is distributed among the cores as follows:

Core A	32 elements
Core B	128 elements
Core C	64 elements
Core D	32 elements

How many time units will be required to process the entire array using the function? Assume that no stalls of any type occur and expressyour answer as N*T. That is specify the integer value for N. Total execution time = max(32/1, 128/2, 64/3, 32/1) = 64 (time units)

b) Ifthe work is distributed among the cores as follows:

Core A	48 elements
Core B	128 elements
Core C	80 elements
Core D	Unused

How many time units will be required to process the entire array using the function? Assume that no stalls of any type occur and expressyour answer as N*T. That is specify the integer value for N. Total execution time = max(48/1, 128/2, 80/3, 0/1)= 64 (timeunits)

6.A program runs on a multi-core system in which each core is rated at 500 MIPS.The program consists of five threads: four independent threads (A,B,C and D) each of which generates partial results. The four sets of partial results are then taken as input by a single thread (E) to produce the final result. Threads A, B, and C each executes 20 million instructions and thread D executes 38 million instructions. Thread E executes 12million instructions in generating the final result. What speedup factor would be provided by increasing the number of identical cores in the system from 2 to 4?

A, B, C and D must complete before E executes.With 2 cores any pair of threads selected from A, B and C could run in parallel and would take time T = 20/500 = 0.04 seconds to complete. The thread not included in the pair would execute in parallel with D which would take 38/500 seconds. Thread E would then take 12/500 seconds for a total of (20+38+12)/500 =70/500 seconds.

Using 4 cores A,B,C and D could each execute on a separate cores,butit wouldtake 38/500 seconds to complete the four threads. Thread E would then take an additional 12/500 seconds for a total of 50/500 seconds. Hence the speedup would be (70/500) / (50/500) =70/50 =1.4.

7.A dual processor SMP system includes an LI data cache for each processor and employs the MESI protocol to maintain cache consistency. Each cache is a direct-mapped copy-back cache that contains 8192 cache lines each of which is 256 bytes in size. A write-allocate policy is used for each cache. One process, P1, runs on the first processor at the same time that another process, P2, runs on the other processor. P 1accesses a variable X with an initial value of 80 that resides in memory at address Ox400800CO. P2 accesses a variable Y with an initial value of 200 that resides in memory at address Ox400800F8.

a) Into which line within Pl 's cache will the memory block containing the variable X be loaded?

The block containing X would map to line Ox400800 MOD 8192 =2048 in Pl's cache

b) Into which line within P2's cache will the memory block containing the variable Y be loaded?

The block containing Y would map to line Ox400800 MOD 8192 =2048 in P2's cache.

c) All of the lines in each processor's data cache are initially invalid. For the following accesses in the order listed, show the MESI state of the affected cache line before and after the reference and explain why any change in state occurs:

Reference	Pl's cache he state before	Pl's cache he state after	P2's cache he state before	P2's cache he state after
Pl increments X by 1	I	M	I	I
P2 multiplies Y by 2	M	I	I	M
Pl reads X	I	S	M	S
P2 reads X	S	S	S	S
P2 increments Y by 3	S	I	S	M
Pl decrements Y by 4	I	M	M	I
P2 multiplies X by 2	M	I	I	M
Pl multiplies Y by 2	{	M	M	I

To increment X, P1must read the block containing X into its cache and modify the line.

To multiply Y by 2, P2 must read the block containing Y into its cache and modify the line.But Pl must first write its copy of the line back to memory. So Pl's copy changes from M to I. P2's copy changes from I to M.

When Pl reads X again, the block containing X must be loaded into Pl's cache.But P2's copy must first be written back to memory and its state changes from M to S. Pl's line changes from I to S.

When P2 reads X, the line in P2's cache containing S remains in the shared state and Pl's copy is unaffected.

When P2 increments Y by 3, its line changes state from Sto M and Pl's line containing the copy changes state from Sto I.

For Pl to decrement Y by the modified line in P2's cache must first be written back to memory and its state changes from M to I. Pl must load the block into its cache and modify the line. So the state of its line will change from I to M.

When P2 multiplies X by 2, its copy of the line changes from I to M after Pl writes its copy back to memory and changes its state from M to I.

Finally, Pl multiplies Y by 2 causing P2 to write its copy back to memory and invalidate its line. The block is loaded into Pl's cache and the state of the line in Pl's cache changes from I to M.

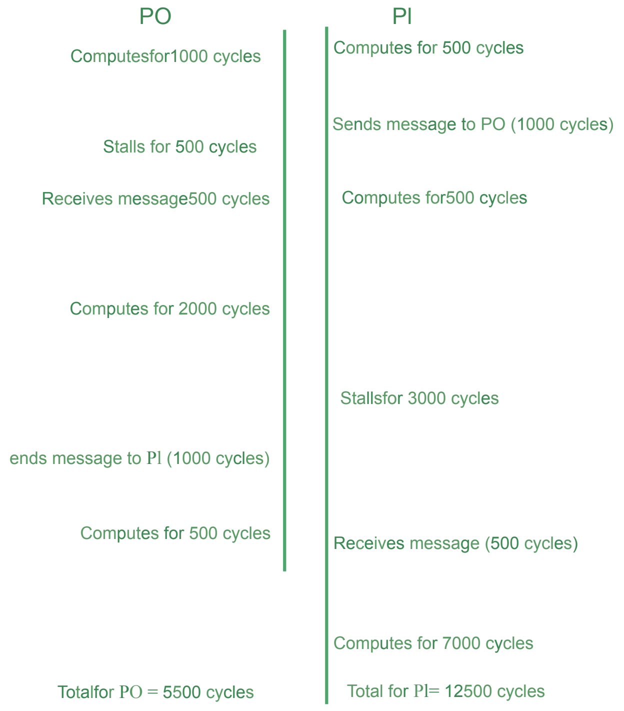
8.A message-passing program runs on two processors within a computer cluster.The time required to send a message is 1000 cycles. It takes 500 cycles to complete the RECEIVE operation once a message is available. The two tasks within the program perform the following actions:

Processor 0
Computes for 1000 cycles
Receives message 1from processor 1
Computes for 2000 cycles
Sends message 2 to Processor 1
Computes for 500 cycles

Processor 1
Computes for 500 cycles
Sends message 1to Processor 0
Computes for 500 cycles
Receives message 2 from processor 0
Computes for 7000 cycles

When a processor sends a message, it waits untilthe transmission completes before continuing . Whatis the total number of cycles that would elapse by the time the program complletes?

Both processors start at the same time. After 500 cycles processorl sends a message to processorO. ProcessorsO attempts to receive the message which arrives after a total of 1500 cycles have elapsed. The RECEIVE completes for processorO after a total of 2000 cycles. At this time Processorl attempts to receive a message. But processorO computes for 2000 cycles before sendingthe message. So the message arrives at processorl after a total of 5000 cycles. After a total of 5500 cycles, processorO completes its task. Processorl completes its RECEIVE after a total of 5500 cycles and computes for an additional 7000 cycles. So processorl completes its task after a total of 12500 cycles.



Module 12 Example set 4

1. What is the difference between strong scaling and weak scaling?
Strong scaling refers to using multiple processors to speedup the execution of a program for a fixed problem size. Weak scaling on the otherhand, increases the number of processors as the problem size increases.

2. Suppose that f is the fraction of the execution time for a program that is due to some operation that can be split into n independent parts. The remaining fraction of the execution time, (1-f), is due to the purely sequential part of the program. Based on Amdahl's law, what is the expresion that gives the speedup for the program provided by using n processors or cores compared to using a single processor or core.

The speedup would be $T1 / T_n$
Where T1 is the execution time using a single processor or core, and
T_n is the execution time using n processors or cores.
According to Amdahl's law $T_n = Ti*[(1-f) + f/n]$
Hence the speedup $= Ti / (T1*[(1-f) + f/n]) = 1 / (1-f) + f/n$

3. What is the upper limit on the speedup for the program, described in problem 2 above, as the parallel part of the program is split into more and more separate independent parts?
The expression f/n gets smaller and smaller with increasing values of n and approaches 0 as n approaches infinity. Hence the speedup approaches $1/(1-f)$.

4. Suppose that it is not possible to subdivide the parallel part of a program into more than 8 independent parallel parts. If the system contains 32 cores, then each of the 8 independent parts would be handled by a separate core. How can the remaining 24 cores be used to advantage?
One way to take advantage of the extra cores is to expand the problem so that after the sequential part of the program is executed, four groups of 8 cores can each be applied to four different datasets as opposed to the single dataset That is, weak scaling can be used.

5. A certain program consists of a sequential part that requires a time s to complete and N parallel parts each of which takes the same time p to complete.

a) Write down an expression for the total time required to complete the entire program using a single processor.
Ti = s + Np
(after completing the sequential part, the processor would execute the N parallel parts one after the other)

b) Write down an expression for the total time required to complete the entire program using N processors.
TN = s + p
since all parallel parts are executed at the same time and each takes time p, the total time is the sum of the sequential time s plus the time p for the parallel parts.

c) Write down an expression for F, the fraction of the total execution time that is accounted for by the parallel part.
The time for the parallel part is p and the total time is s + p.
So F = p/ (s + p).

d) Write down an expression for the speedup provided by using N processors as a function of F (the fraction of the total execution time that is accounted for by the parallel part).
Speedup = Ti / TN = (s + Np)/ (s+p) = s/(s+p) + N* p/(s+p)
Part c) above shows that F = p / (s+p)
Hence 1 - F = 1 - p/ (s + p) = [(s+p) - p] / (s+p) = s / (s+p).
Therefore speedup = 1 - F + N*F

This is known as Gustafson's law.

6. Auniprocessor system accesses memory over a 32-bit bus. The processor has a write-back (as opposed to write through) data cache employing a write-allocate policy. The data cache ooperates in look-through mode, uses true LRU replacement and contains 65536 lines each of which is 1024 bytes in size. Each data cache access takes 2 CPU clock cycles to either detect a cache miss, or to perform a cache read or to perform a cache write. Loading a memory block into cache or writing a cache line back to memory takes 400 CPU clock cycles. Recall that for a cache miss the data item that is needed is read in parallel with loading the memory block containing the data item into cache. The operating system flushes the contents of cache to memory after a program terminates.

The code shown below reads and updates each of the 134217728 four-byte elements in an integer array. The address of the array in memory i Ox1008A400

```
        lui    $8,0x1008     # point to first element to be processed
        ori    $8,0x,A400
        lui    $4,0x0800     # number of elements = 134217728 (=Ox08000000)
loop:   lw     $12,0($8)     # get next element
        addi   $4,$4,-1      #  decreme nt loop control variable
        sub    $12,$0,$12    # negate by subtracting from 0
        addi   $8,$8,4       # point to next element
        sw     $12,-4($8)    # update element that was read
        bgez   $4,loop       # repeat if more eleme nts remain
        nop
```

a) What would be the data cache hit ratio for this code running on the uniprocessor system? Assume that the data cache is initially empty.

Since the line size is 1024 bytes, each line holds 256 elements. The data cache can hold 65536*256 = 16777216 array elements (1/ 8th of the 134217728 total). The code steps sequentially through the array elements and causes a data cache miss on the initial reference to the first element in each line. The 256 elements in a line are referenced twice (a read followed by a write). Hence for each line there will be 2*256 = 512 references and one miss (i.e., 511 hits and 1 miss). The entire array corresponds to 134217728/ 256 = 524288 lines.
So the data cache hit ratio = (524288*511) / (524288*512) = 511/512 = 0.998 or 99.8%.

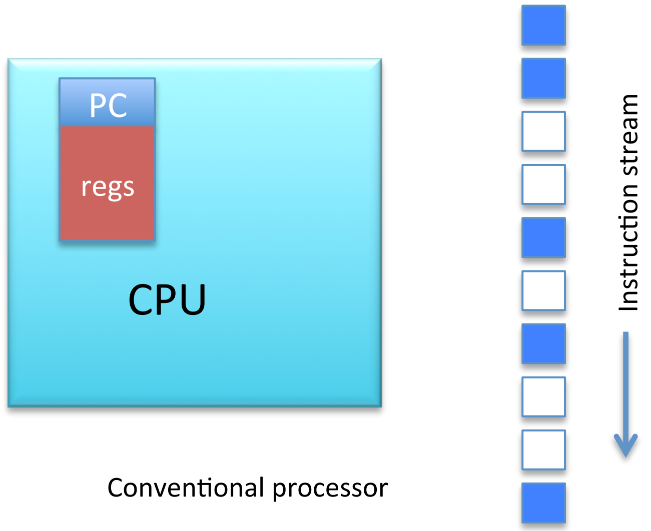
b) Aseparate copy of this code is executed on each of the cores in an 8-core system. Each core has a separate data cache that is identical to that of the uniprocessor system and the code on each core uses the appropriate starting address and loop limit that would correspond to 1/8th of the array. What would be the data cache hit ratio for the code running on each of the 8 cores? Assume that all data caches are initially empty.

The code on each core accesses its 1/8th of the array (524288/8 = 65536 lines) sequentially. So the data cache hit ratio for each core = (65536*511)/(65536*512) = 511/512 = 0.998 or 99.8%. This is the same as the data cache hit ratio seen by the uniprocessor.

Example Set 5

1. How is multi-threading handled on a conventional processor?

A conventional processor is one with a single register set that is owned or used by one thread at a time. It contains a single PC (program counter) that is used to fetch instructions from a single stream at a time.



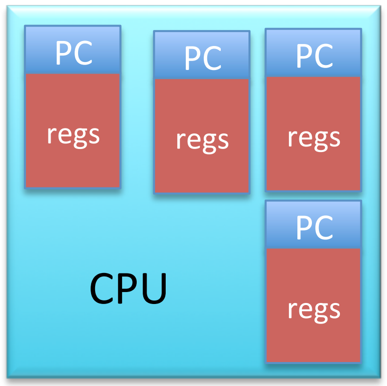
Conventional processor

With superscalar processors the lack of ILP (instruction level parallelism) can cause some execution units to be underused.

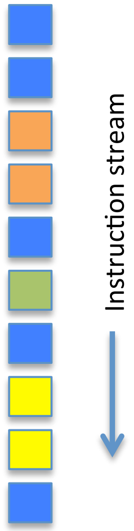
Only one thread can be executing at a time. When the active thread becomes blocked due to, say, a page fault or due to having to wait for an I/O operation to complete, the thread's "context" (i.e., the contents of its registers, its PC, status register, etc.) must be saved to memory and the context of another thread that is ready and waiting to execute must be copied from memory into the CPU's registers.
These context switches usually involve executing operating system code and can take thousands of CPU cycles. If there are no ready threads available, the CPU will idle.

2. How does a multi-threaded processor differ from a conventional processor?

A multi-threaded processor is able to follow multiple streams of execution without the need for software context switches. It employs hardware multi-threading as opposed to software multi-threading.



Multi-threaded processor



Multi-threaded processors can compensate for the lack of ILP by allowing execution resources not used by one thread to be used by another thread. That is, they employ TLP (thread level parallelism).

The multi-threaded processor contains multiple PC's, register sets, etc. which constitute a hardware context. The number of hardware contexts defines the number of threads that can be supported without software intervention.

Since the hardware contexts reside within or near the processor, a context switch can be performed in just a few cycles by simply switching to a different set of registers (including a different PC, status register, etc.).

However the threads share the same memory, branch predictor, cache and TLBs.

3. Often moving companies try to lower cost and increase profits by combining multiple loads within a single moving van rather than sending a partially filled van cross country. How is this analogous to a multi-threaded processor?

A single multi-threaded processor increases throughput by supporting in hardware a number of threads that run concurrently so as to take greater advantage of the available execution resources and to avoid wasting CPU cycles when the CPU becomes idle or when extra cycles are used for software supported context switching. This is analogous to avoiding the waste of unused space in the moving van.

4. Can it be said that hardware multi-threading "virtualizes" the processor?

Yes, hardware multi-threading makes a single-core processor appear to software as a multi-processor. Each thread running on a separate hardware context appears to be running on a separate virtual core that has the hardware capabilities of the physical CPU minus the resources used by the other threads.

5. As processor performance and speed continue to increase, why has hardware multi-threading become more important?

The difference between the time required for internal CPU operations and the time required to access memory on earlier slower processors was not as dramatic as it is for more modern processors. The use of cache memory helped to decrease this difference. However for the much faster modern processors, even a simple cache miss (not to mention waiting for I/O) can incur many hundreds or thousands of cycles as a penalty (due to the need to access the next level in the memory hierarchy to resolve the cache miss). The ability to quickly switch from one thread to another hides these penalties or latencies by allowing other threads to use the idle resources.

6. What are the main models of hardware multi-threading?

Three of the main models are course-grain multi-threading (CMT), fine-grained multi-threading (FMT), and simultaneous multi-threading (SMT).

Course-grained multi-threaded processors execute one thread at a time, but employ hardware contexts to switch from one thread to another in just a few cycles to hide long latencies (i.e., delays) such as memory accesses.

Fine-grained multi-threaded processors can context switch every cycle with no delay. They interleave instructions from different threads on a cycle by cycle basis. However, during any given cycle, they only issue instructions from a single thread.

Simultaneous multi-threaded processors can issue instructions from multiple threads in the same cycle. This allows the issue width (i.e., the number of available execution units) to be more fully used. The issue width is also known as the superscalar degree. This takes advantage of the issue width even when one thread does not have enough ILP to fully use all of the available execution units. Units that are not required by instructions from one thread may be needed by instructions from another thread.

7. List some of the possible sources of latencies or delays that can be experienced by modern processors.
The sources of latency include data cache misses, memory latencies, instruction cache misses, instruction dependencies, branch miss-predictions, TLB misses, and communication delays between processors.

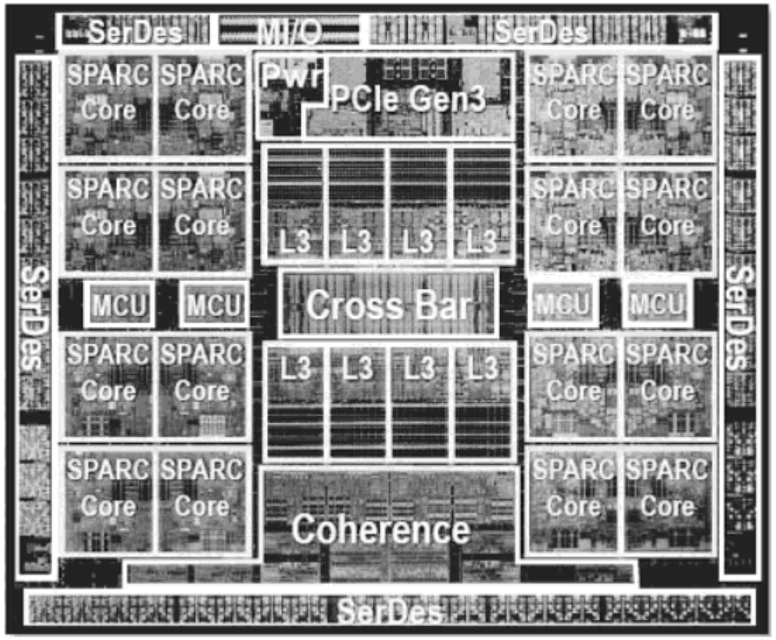
Example Set 6

1. What is a multi-core architecture?

A multi-core architecture is one in which multiple processor cores are integrated on a single die (i.e., chip).

A processor core or CPU is a processing element capable of independently fetching and executing instructions from at least one instruction stream. The core typically includes logic such as an instruction fetch unit, a program counter, instruction scheduler, functional units, register file, etc.

An example of a multi-core system (the Oracle T5) with 16 cores on a single die is shown below:



2. What is a parallel architecture computer?

A parallel computer is a collection of processing elements that communicate and co-operate to solve a large problem fast.

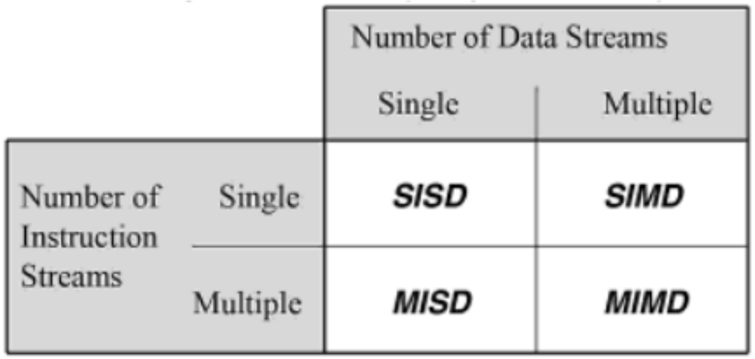
The processing elements can be CPU's where each CPU is a separate chip or they may correspond to separate cores on a single chip. Early parallel computers employed separate CPU chips because the density of transistors on chips was relatively low compared to more recent modern systems.

3. What is the main characteristic of a vector processor or core?

The main characteristic is that the core applies the same operation to multiple data items simultaneously. It fetches instructions using a single program counter. For example, a vector addition instruction reads elements from two arrays and performs a pairwise add of elements from the two arrays and writes the sums into a third array.

4. What are the categories defined by Flynn's taxonomy of parallel computers?

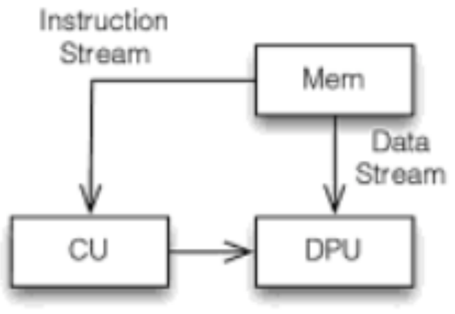
They are SISD, SIMD, MISD and MIMD as shown below:



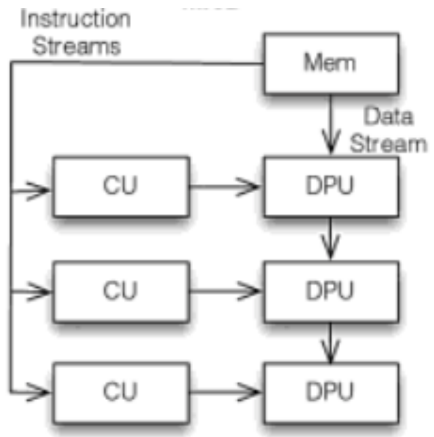
		Number of Data Streams	
		Single	Multiple
Number of Instruction Streams	Single	***SISD***	***SIMD***
	Multiple	***MISD***	***MIMD***

5. Label each of the diagrams below, as a SISD, SIMD, MISD or MIMD system. “CU” means control unit and “DPU” means data processing unit.

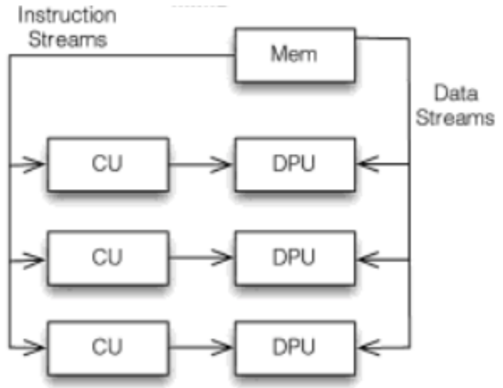
a)



b)



c)



d)

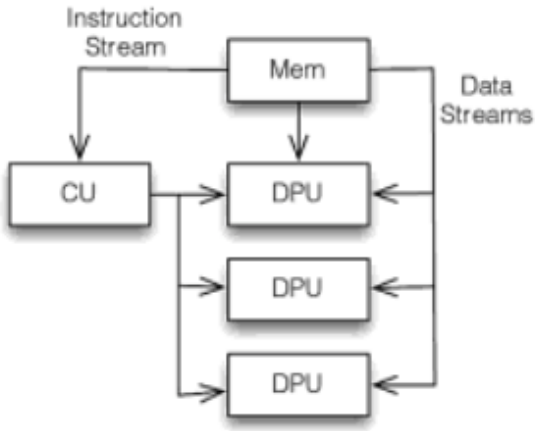
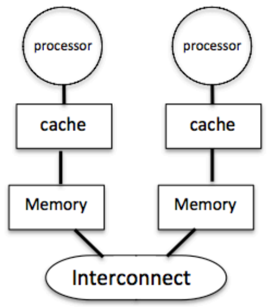
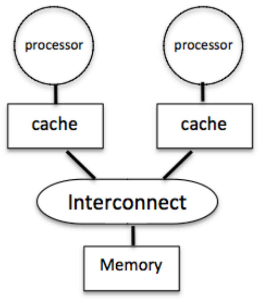


Diagram a) is corresponds to SISD, b) corresponds to MISD, c) corresponds to MIMD and d) corresponds to SIMD.

6. Each of the following diagrams represents a version of a MIMD system. Label each one as UMA, NUMA or NORMA.

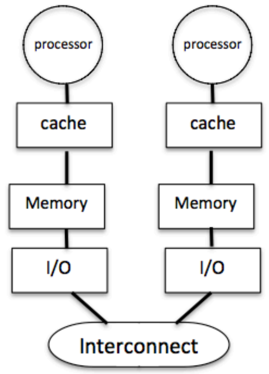
a)

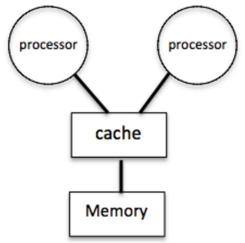




b)

c)





d)

a) NUMA (non-uniform memory access) Accessing memory via the interconnect takes longer than not using the interconnect.

b) UMA (uniform memory access) Each processor takes the same time to access memory.

c) NORMA (no remote memory access, also called a cluster or distributed processing system). Each processor must issue a request to access the memory owned by a remote processor. It can only directly access is own local memory.

d) UMA (uniform memory access with a shared cache). The processors take the same amount of time to access a shared memory and cache.

7. What are some of the factors that motivated the switch to multi-core processors?

Over the years a number of techniques and approaches have been used to increase the performance of single processors. These include pipelining and superscalar operation. The faster switching speeds and higher densities of transistors that can be implemented on a chip made it possible to greatly increase the processor's clock rate. It also made it possible to implement the complex logic needed to manage the dynamic instruction scheduling required for the out-of-order issue and completion of instructions.

The time and effort required to design, test and verify these complex logic systems grew to the point that incremental improvements in the performance became too costly for many systems.

The power consumed by such high-speed logic and the heat produced became a limiting factor. In some cases the power consumption and required heat dissipation increased by a factor of 100 or more.

Multi-core (also called chip multiprocessor or CMP) systems offered an alternative use for the higher transistor densities. Instead of using the logic to implement sophisticated highly pipelined dynamic instruction scheduling superscalar systems, multiple copies of processors with simpler designs that took less time and effort to design and verify were simply stamped onto a single chip. These "cores" run at lower clock rates, have pipelines with fewer stages, and do not require special cooling to keep them operating correctly. Comparable throughput and performance can be achieved with these multi-core or CMP systems as with the more costly high-end single processor superscalar systems.

Example Set 7

1. What does it mean when a cache line is referred to as "clean"?
When a memory block is loaded into some cache line, the line is said to be clean because its contents matches that of the memory block. Once the cache line is written to it is no longer clean (i.e., it is dirty).

2. What is the purpose of the dirty bit in a uniprocessor write-back cache system?
Each cache line has a bit used to record whether the contents of the line has been altered. This "dirty bit", when set, indicates that the cache line must be written back to memory when the line is removed from the cache. Lines whose dirty bit is not set match the original contents of the block in memory and need not be written back upon removal from the cache.

3. In a shared memory multiprocessor system in which each processor has a separate cache, what does it mean for a cache line to be in the "modified" state?
If a line in a cache is in the modified state, this means that the line is not only dirty (i.e., has been altered) but also that there are no other caches that contain a copy of the same line.

4. What is the difference between a "write invalidate" strategy and a "write update" strategy for a cache?
Both strategies describe what happens when a processor in a multiprocessor system writes to a line in its cache. With the write update strategy all copies of the line in any other caches are also updated. With the write invalidate strategy all copies of the line in any other caches are marked as invalid.

5. What is an advantage of a write invalidate strategy and of a write update strategy?
Write invalidate is advantageous when a processor tends to write repeatedly to the same line because only one invalidation needs to occur. However if the copies of the line in other caches are invalidated, attempts to read the line by other processors would result in misses and would require reading from the much slower memory. So the write update strategy would be advantageous when a write to a line by a processor is followed by reads that map to the same memory block by other processors.

Module 12 Example Set 1

1. How does a process differ from a thread?
A program that has been loaded into memory and is able to execute is referred to as a process. Each process has an address space determined by the width of the PC register (e.g., a 32-bit PC corresponds to a 4GB address space). The address spaces of different processes are separate and distinct. One process cannot access items within the address space of another process. A process contains one or more threads of execution. Each thread within a process has a separate context which defines the state of the thread. The threads within a process share a common address space and can communicate via the shared address space while separate processes can only communicate via files or with the assistance of the operating system. Threads are sometimes referred to as "lightweight processes".

2. What defines the context or state of a thread?
The state of a thread is primarily composed of the value in the program counter (PC), the contents of the general purpose CPU registers and the contents of special purpose registers such as memory management registers or program status registers. A thread that has been previously stopped or suspended can later be resumed with no adverse effects if its state (i.e. context) is saved and restored properly.

3. How is multi-threading handled on a conventional uniprocessor system?
A conventional processor only executes instructions from one thread at a time. If the active thread has to stall or wait for an event such as a page fault or I/O request to be processed, the operating system will save the context (all of the registers associated with the thread) in memory (for example, on a stack) and will fill the CPU registers with the information (i.e. the state) of another thread that the OS selects to be executed next. The software that performs a "context switch" can take thousands of CPU cycles.

4. What is meant by a "multi-threaded architecture"?
A multi-threaded architecture is one in which a single processor has the ability to execute multiple streams of instructions without the need for software supported context switches. The CPU register set is replicated to provide the abililty to quickly (in just a few cycles) switch from one hardware context to another without software intervention. A separate set of registers is available for each thread.

5. How do superscalar and VLIW systems differ from multi-threaded processors?
Superscalar and VLIW processors are single-threaded because they execute programs using a single PC (program counter) and a single register file. Multi-threaded processors, on the other hand, have multiple program counters and register files each of which can be used to fetch and execute instructions from different threads at a time. Superscalar and VLIW systems issue multiple instructions in a single cycle, but these instructions all come from the same thread. Multi-threaded processors can execute instructions from multiple threads in the same cycle. The available execution units can be shared between threads in a multi-threaded system.

6. What are the different styles or types of multi-threading?
There are three major types:

I. Coarse-grained – in which a processor runs a single thread until some long latency event such as a cache miss triggers a switch to a different thread. The instructions from the stalled thread are drained from the pipeline and instructions from the newly activated thread are fed into the pipeline.

II. Fine-grained – the processor is switched from one active (i.e. not stalled) thread to another on every clock cycle (the instructions from different threads are interleaved), but only instructions from one thread are issued during any particular cycle.

III. Simultaneous multi-threading – in which instructions from multiple threads are issued during the same cycle to any available execution units that they require within a superscalar processor.

7. How can instructions from different threads be handled within a single pipeline without causing data dependencies or interferring with each other?
Multiple separate register sets (one for each thread) have to be used so that instructions from one thread do not overwrite the registers used by other threads.

8. With a multi-processor system, independent threads executing in parallel must run on separate processors. Assume a parallel program containing three threads (thread1, thread2 and thread3) is to be executed on a multi-processor system that contains 2 processors. Thread 1 takes T time units, thread 2 takes 2T time units and thread 3 takes 3T time units.

a) How many time units are required for the entire program if threads 1 and 3 are executed on the two processors, followed by executing thread 2 on one of the processors?
Executing threads 1 and 3 in parallel would take 3T time units (the time required for the longer of the two threads). Then thread 2 would take 2T time units; so the total would be 3T + 2T = 5T time units.

b) How many time units would be required for the entire program if ony one of the processors was available?
The three threads would have to execute sequentially one after the other, so the total time would be 1T + 2T + 3T = 6T time units.

c) What would be the best order in which to execute the three independent threads?
This would correspond to the shortest total time which is achieved by executing threads 2 and 3 in parallel. Thread 2 would complete after 2T time units. The processor that ran thread 2 could then be used to execute thread 1 in parallel with the completion of thread 3. Hence after 3T time units all three threads would be completed. After the first 2T time units, thread 2 would finish and thread 3 would complete its first 2T time units. In the next T time units threads 3 and 1 would complete.

9. What is meant by "barrier synchronization"?
Separate processes or threads when colaborating on a particular problem may have to synchronize their activity at strategic points. One way to do this is to have each process or thread execute up to a certain point within the code and wait for all of the others to reach the synchronization point after which they all proceed. This technique is referred to as barrier synchronization.

10. How might sharing a common address space cause dependencies between threads?
A result generated and stored into memory by one thread may be required by one or more other threads as an input. Often semaphores or mutexes must be used to prevent threads from accessing a shared memory operand at the same time. Semaphores or mutexes can be thought of as flags that can be set or cleared to indicate whether a shared item is in use or is free.

11. According to Amdahl's law as it applies to multi-processor systems, what effect does increasing the number of processors have on the execution time for a fixed size program consisting of a sequential part and a parallel part that can be split among multiple processors?
The sequential part would execute on a single processor and only the parallel part of the program would benefit from the use of the additional processors. Hence as the number of processors is increased, the parallel part would take a smaller and smaller fraction of the total execution time.

Module 12 Example Set 2

1. What are chip multiprocessors (CMPs) and how do they differ from multi-threaded systems?
Chip multi-processors (CMPs) are also known as multi-core processors. They are examples of one version of multi-threaded systems. CMPs can execute instructions from multiple threads at the same time, but on distinct cores. No execution units are shared among the threads. Each core has a separate L1 cache and separate execution units but may share the L2 or L3 cache and the memory system.

3. How do multi-core systems differ from multi-processor systems?
Each processor within a multi-processor system is implemented on a separate chip and employs separate caches while interfacing with other processors through some type of interconnect system that allows the separate processors to share memory. The cores within a multi-core system are on the same chip and tend to be simpler in nature than the multi-processor's more complex superscalar processors that provide dynamic instruction scheduling. Each core usually has a separate L1 cache but may share the L2 and L3 caches with other cores but the processors within a multi-processor system would have separate caches that must be kept consistent or coherent.

4. What are some of the factors that have prompted the development and use of multi-core processors?
The management of reservation stations and the dynamic scheduling of instructions within superscalar systems together with the control of very deep pipelines (20 or more stages) require the use of a vast number of transistors which when operated at extremely high clock rates consume large amounts of energy and produce a lot of heat which becomes more of a problem to handle. Also the design time and test/verification time for such systems adds dramatically to their cost. So, the use of multiple cores, each of which has a much simpler design and runs at a lower clock rate, has become viewed as a better and more cost effective alternative to complex powerful single core systems. It has also been observed that most programs often do not have enough instructions that can be executed in parallel due to various types of dependencies and due to cache misses which force accesses to the relatively slow memory. So the sophisticated superscalar systems are often starved for work to do and many of their functional units tend to sit idle. Implementing multiple cores on a single chip often takes no more chip area or real estate than a single complex superscalar processor.

5. Describe a typical application of a multi-core system.
One example is a web server system that needs to handle a large number of requests over a network. The individual requests, whether for web pages, database access or file service, are typically independent tasks that can be handled by threads spread across separate cores or processors.

6. How does the type of parallelism exploited by multi-core systems differ from that exploited by superscalar systems?
Each of the separate cores within a multi-core system executes a separate thread within a program or from different programs at the same time; this is called thread-level parallelism (TLP). Superscalar systems, on the other hand, dynamically examine the instructions within a single program or thread and try to identify multiple instructions that can execute at the same time; this is called instruction level parallelism (ILP). The superscalar systems often execute the instructions out of order (i.e., in a different order than they appear within the program), but insure that the instructions write their results in the proper order using a re-order buffer (ROB).

7. How are a processor's clock rate and voltage level related to its power consumption?
The power consumed is proportional to the clock rate and to the square of the voltage level used.
Systems that run at lower clock rates can usually use lower voltage levels as well. For example, if cutting the clock rate in half also allows the voltage to be cut in half, then the overall power consumption will be cut by a factor of 8 (since ½ * ¼ = 1/8).

8. Do multi-core processors tend to execute programs at a faster rate than superscalar uniprocessors running at comparable clock rates?
No, the individual cores usually do not execute as many instructions in parallel as the heavyweight superscalar systems. However, the multi-core systems tend to greatly increase the number of threads or tasks that can be completed per unit time (i.e., the throughput) and provide a higher performance per watt advantage.

9. How does a vector processor differ from a scalar processor?
A vector processor contains vector registers and has the ability to perform the same operation on the elements in two vector registers at the same time.
Scalar processors can only apply one operation at a time using two scalar (i.e., single element) operands as inputs.

10. What are the options for loading vector registers?
One option is to load all elements within the vector registers before the operation begins, the other to employ "chaining" in which the first element in each input vector register is loaded and while each subsequent element is loaded, the operation is applied to the previously loaded elements (in a pipelined fashion).

11. A double precision floating point vector A of length 64 (i.e., A is a single dimensional array containing 64 elements) with stride 2 resides in memory at address 0x8c040020. What would be the memory address of elements A[3] and A[9]?

The “stride” is the separation between consecutive elements. A stride of 1 means that consecutive elements are adjacent in memory, A stride of N means that consecutive elements are separated by N*W bytes, where W is the width of each element. Since each element is 8 bytes in size, the distance between consecutive elements 2*8 = 16 bytes (0x10). The address of A[0] is 0x8C040020, the address of A[1] is 0x8C040020 + 1*2*0x10,
so the address of A[3] is 0x8C040020 + 3*2*0x10 = 0x8C040080.
The address of A[9] = 0x8C040020 + 9*2*0x10 = 0x8C040140.