Answers:

1.

(a) Heavy multicore processors:
In heavy multicore processors, the multiple cores on a chip requires inter-core communication mechanisms. The individual processors use a common bus which is shared by all processors to communicate in a shared memory multiprocessor. It was followed by general purpose vendors such as AMD or Intel. The designs of recent processors are based on the realization that the buses which are shared communication mediums interferes the bandwidth and latency. It is known that a multiprocessor is sequentially consistent if the outcome of any execution is the same as that of the memory operations carried out by all threads in a sequential order.

(b) Lightweight cores with SoC characteristics: memory controllers and I/O interface on die:
The lightweight cores with SoC characteristics are designed on the basis of strict requirements of application domains in terms of flexibility and processing performance. The new emerging designs with low cost are based on SoC platforms which can be hardware or software. These platforms integrate multiple processors core as a programmable resource with hardware accelerators on a single chip that is cost efficient. Programmability is introduced in these single chip architectures. These components function for specific domains like telecommunication and multimedia, signal processing and many more.

(c) GP-GPUs:
A GPU i.e. Graphical processing unit is a single chip processor used for 3D applications that transforms objects when a 3D scene is rendered. It renders images, animations and video for the computer's screen. In early stage, the CPU performed all these functions which occupied more cycles and made the functioning slower. Thus, GPUs were invented to offload those tasks from CPU and free up its processing power by reducing the strain. Now, the GPUs are integrated with CPU on the same circuit in some machines. Some of the companies that manufactures best GPUs are NVIDIA, Intel, AMD and ARM.

Cont. on Next Page.

Comparative Study:

	Number and complexity of their processing units	**Memory Hierarchies**	**Communication Interconnections**	**Application domains**
Heavy Multicore processors	The number of processing units are high but lower than that of GPU	DRAM with high bandwidth	Higher frequency for fast access of memory and I/O	Desktops, laptops, servers, big data analytics
Lightweight cores with SoC characteristics	Number of processing units per core are lower	On chip caches and Low power	Higher bandwidth at lower energy. It avoids bottleneck situations	Devices like sensors, smart watches, smart phones because they use low power.
GPUs	In GPU, the number of processing units are higher than both Heavy multicore processors and Lightweight cores with SoC	Uses main memory and connects buses with high speed	High bandwidth and performance is optimized. To avoid bottleneck, it uses faster speed interconnections.	Gaming platforms, data intensive clusters, video games

2.

(a) We know that Amdahl's law is defined as: $Speedup = \frac{Exeution\ time\ (old)}{Execution\ time\ (new)}$.

Here, in the case of this multiprocessor, we need to calculate the new execution time in order to compute the formula for speedup.
It is given that for an application, we have a function of the form 'f(i, p)' and it gives the fraction of time where 'i' is the number of processors that are usable and 'p' is the total number of available processors. It means that for the portion of the original execution time that can use 'i' processors is given by 'f(i, p)'. Thus, $\sum_{i=1}^{p} f(i,p) = 1$, i.e. 100%.
Now, if we assume that the Execution time (old) is 1, then for 'p' processors the relative time of the application is the addition of the times required for each portion of the execution time that can be increased by using 'i' processors. Here, 'i' is between 1 and 'p'.
Thus, Execution time (new) = $\sum_{i=1}^{p} \frac{f(i,p)}{i}$.
We use this value of Execution time (new) in the formula for Speedup in the following way: $Speedup = \frac{1}{\sum_{i=1}^{p} \frac{f(i,p)}{i}}$. This makes the Amdahl's law a function of the available 'p' processors.

(b) For 8 processors, the speedup can be calculated as:

$$Speedup = \frac{1}{Execution\ time\ (new)} = \frac{1}{\sum_{i=1}^{p} \frac{f(i,p)}{i}} = \frac{1}{\sum_{i=1}^{8} \frac{f(i,8)}{i}}$$

$$= \frac{1}{\frac{f(1,8)}{1} + \frac{f(2,8)}{2} + \frac{f(4,8)}{4} + \frac{f(6,8)}{6} + \frac{f(8,8)}{8}}$$

$$= \frac{1}{\frac{0.2}{1} + \frac{0.2}{2} + \frac{0.1}{4} + \frac{0.05}{6} + \frac{0.15}{8}}$$

$$= \frac{1}{0.2 + 0.1 + 0.025 + 0.0083 + 0.018}$$

$$= \frac{1}{0.3513}$$

Speedup = 2.846

(c) For infinite number of processors, 128 processors can handle total 100% of the fraction of time. Thus,

$$Speedup = \frac{1}{Execution\ time\ (new)} = \frac{1}{\sum_{i=1}^{p} \frac{f(i,p)}{i}} = \frac{1}{\sum_{i=1}^{\infty} \frac{f(i,\infty)}{i}}$$

$$= \frac{1}{\frac{f(1,\infty)}{1} + \frac{f(2,\infty)}{2} + \frac{f(4,\infty)}{4} + \frac{f(6,\infty)}{6} + \frac{f(8,\infty)}{8} + \frac{f(16,\infty)}{16} + \frac{f(128,\infty)}{128}}$$

$$= \frac{1}{\frac{0.2}{1} + \frac{0.2}{2} + \frac{0.1}{4} + \frac{0.05}{6} + \frac{0.15}{8} + \frac{0.2}{16} + \frac{0.1}{128}}$$

$$= \frac{1}{0.2 + 0.1 + 0.025 + 0.0083 + 0.018 + 0.0125 + 0.00078}$$

$$= \frac{1}{0.3645}$$

Speedup = 2.743

3.

(a) To check if there is loop carried dependencies or not, we use GCD test.
If GCD(2,4) divides 5-4, then there is a dependency.
Now, GCD(2,4) = 2 and (5-4)%2 = 1. It means GCD(2,4) does not divide 5-4. Therefore, for B[], there is no loop carried dependencies.
We use same index for A[] in the body of the loop, thus no loop carried dependencies exist for A[].

(b) In the given loop:
True dependencies are: S1 and S2 lines because of A[i]
S3 and S4 lines because of A[i]

Output dependencies are: S1 and S3 lines through A[i]

Anti-dependencies are: S1 and S2 lines due to B[i]
S2 and S3 lines due to A[i]
S3 and S4 lines due to C[i]

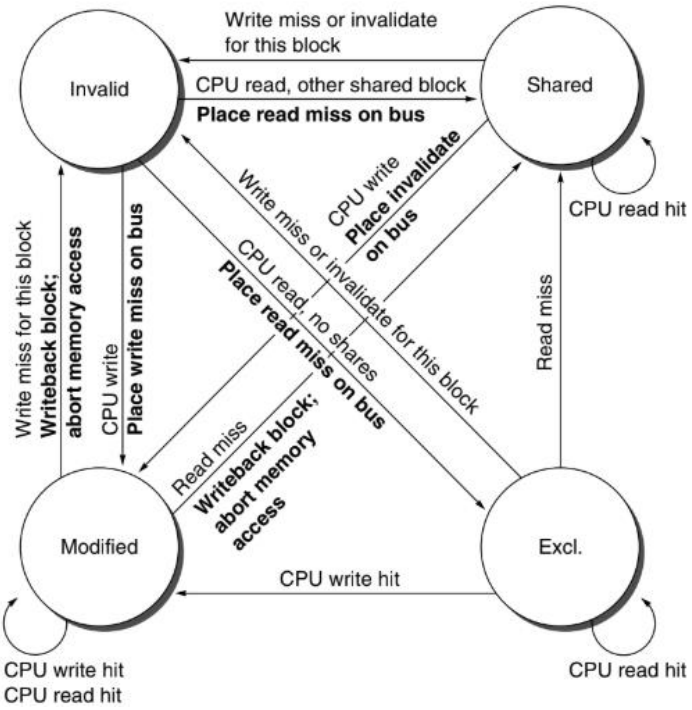
Eliminating output dependencies and anti-dependencies by renaming:
Assuming arrays A1, B1 and C1 as the copies of A, B and C in the given loop. Considering an array X[].

```
for (i=0; i<100; i++)
{
        X[i] = A1[i] * B1[i];  /* S1 */
        B[i] = X[i] + c;       /* S2 */
        A[i] = C1[i] * c;      /* S3 */
        C[i] = D[i] * A[i];  /* S4 */
}
```

This above code will remove output dependencies and anti-dependencies.

4.

(a) The new protocol diagram for a MESI protocol that adds the Exclusive state and transitions to the base MSI protocol's Modified, Shared and Invalidate states is given below:



(b) The total stall cycles for the following code sequences with both the base protocol and the new MESI protocol are:

i. C0: R, AC00, Read miss, Satisfied in memory, No sharers MSI: S, MESI: E

C0: W, AC00 <- 40, MSI: Send Invalidate, MESI: Silent transition from E to M

For MSI, total stall cycles = 100 + 15 = 115 stall cycles

For MESI, total stall cycles = 100 + 0 = 100 stall cycles

ii. C0: R, AC20, Read miss, Satisfied in memory, Sharers both to S

C0: W, AC20 <- 60, Both send invalidates

For both MSI and MESI, total stall cycles = 100 + 15 = 115 stall cycles

iii. C0: R, AC00, Read miss, Satisfied in memory, No sharers MSI: S, MESI: E
C0: W, AC20, Read miss, Memory, Silently replace AC20 from S or E
For both MSI and MESI, total stall cycles = 100 + 100 = 200 stall cycles, silent replacement from E.

5. We know that it is a 64-processor distributed-memory multiprocessor.

Clock Rate = 2.0 GHz

Base CPI = 0.75

Instructions involving a remote communication reference = 0.2%

Cost of a remote communication reference = (100 + 10h) ns, where 'h' is the number of communication network hops.

(a) The worst-case remote communication cost for 64 processors arranged as:

A Ring:

Largest number of communication network hops = 32

Cost = (100 + 10(32)) ns = 100 + 320 = 420 ns

An 8*8 processor grid:

Largest number of communication network hops = 14

Cost = (100 + 10(14)) ns = 100 + 140 = 240 ns

A hypercube:

Largest number of communication network hops = ($\log_2$ 64) = 6 (because longest communication path on a 2^n hypercube has 'n' links)

Cost = (100 + 10(6)) ns = 100 + 60 = 160 ns

(b) We are given that the Base CPI = 0.75

Comparing the CPI with the 64 processors arranged as:

A Ring:

Worst-case CPI = 0.75 + (0.2% of communication cost) = 0.75 + ((0.2/100) * 420)

= 0.75 + 0.84 = 1.59 CPI

An 8*8 processor grid:

Worst-case CPI = 0.75 + (0.2% of communication cost) = 0.75 + ((0.2/100) * 240)

= 0.75 + 0.48 = 1.23 CPI

A hypercube:

Worst-case CPI = 0.75 + (0.2% of communication cost) = 0.75 + ((0.2/100) * 160)

= 0.75 + 0.32 = 1.07 CPI

6.

(a) It is given in the numbers from textbook that the baseline WSC used 45,978 servers. Here, if the server operator decided to save power costs by running all servers at medium performance, then the performance is 75% of the original.
Thus, to achieve the same level of performance, the number of servers required can be calculated as: 0.75 * number of servers required = 45,978
Thus, number of servers required = 45,978 / 0.75 = 61,304 servers.

(b) The cost of baseline per server in the textbook section is given as $1,450 and the server cost of such a configuration remains the same as that of baseline, i.e. $1,450. According to the table given, for medium performance, the power per server becomes 60% of the baseline. It means here, Power per server = 0.6 * Power of baseline = 0.6 * 165 W = 99 W.
We can figure the CAPEX numbers using the above information as:
Network = $16,444,934.00
Server = $88,890,800.00
Now, we can calculate the total critical load using the following formula:
Critical Load = Number of servers * Watts per server + Network load
It is given that he Network Load is 531,483
Thus, here the Critical Load = 61,304 * 99 W + 531,483 = 6,600,579 W.
We can calculate the OPEX numbers using the calculated Critical Load along with the servers. We get the OPEX numbers as:
Power = $391,256
Total = $4,064,193.

(c) If a server that is 20% cheaper is bought, the server cost is 0.8 * Cost of baseline per server = 0.8 * $1,450 = $1,160. We know that the cheaper server is x% slower, thus the number of servers required is (Number of servers used by baseline) / (1-x) = 45,978 / (1-x).
After knowing the number of servers required, we can calculate the network components using the spreadsheet.
As we know, Critical Load = Number of servers * Watts per server + Network Load.
Also, it is given that the cheaper server uses y% less power. Thus, Watts per server = (1-y) * Watts per server used by baseline = (1-y) * 165 W.
Thus, using the information calculated above and attaching them into the spreadsheet, we can build the performance-power curve that provides a TCO comparable to the baseline server where it turns out that they are approximately equal at some points.

7. A GPU is given that has 8 multiprocessors with a 2 GHz clock.
Given: Each processor has 8 multithreaded floating-point units and integer processing units.
8 partitions of 1 GHz Graphic DDRAM
Each Partition Width = 8 byte
Each Partition Capacity = 256 MB
To calculate: Time taken by computation F = A * B where A, B and F are n*n matrices and 'n' is limited by the amount of system memory
Assuming that, GPU has a single precision Floating-point Multiply-Add instruction.
Single precision Floating-point Multiply-Add performance can be calculated as:
Single precision Floating-point Multiply-Add performance = Number of multiprocessors * Number of single precision floating-point units * (Flops/instruction/SP) * Memory system frequency * System frequency

Thus, Single precision Floating-point Multiply-Add performance = 8 * 8 * 2 * 1 * 2 GHz = 256 GFlops/second.

DDRAM memory size = Each partition width * Each partition capacity
= 8 * 256 MB = 2048 MB

Maximum bandwidth = Number of partitions * Bytes/transfer * Transfers/clock * Clock Frequency = 8 * 8 * 2 * 1 GHz = 128 GB/second.

For 32-bit single precision computer system, we assume that if 3 n*n single precision matrices are required, then maximum 'n' can be found by using: $3n^2 + 4 <= 2 * 1024 * 1024 * 1024$
Thus, n = 13377
Number of operations in matrix multiplication (n*n):
Each element 'n' Multiply-Add
Each row n*n Multiply-Add
So, for result, n*n*n Multiply-Add
It gives 2393 GFlops

Assume that we have no cache, then we need to load 2 matrices and 1 on graphics memory.
Thus, in total we need to load $3n^2$ = 512 GB data.
We calculated maximum bandwidth of DDRAM which is 128 GB/second, thus time taken to load 512 GB data can be calculated using: Total data / Maximum Bandwidth
= 512 GB / 128 GB/second = 4 seconds.

Cont. on Next page

The processing time is calculated using following formula:
Number of GFlops / Single precision Floating-point Multiply-Add performance
= 2393 GFlops / 256 GFlops/second
= 9.347 seconds

Thus, the total time taken by matrix multiplication i.e. total time taken to perform the computation F = A * B is addition of time taken to load data and processing time.
Total time taken for the computation 'F = A * B' = 4 sec + 9.347 sec = 13.347 seconds.

References:

1. Textbook: Computer Architecture – A Quantitative Approach by John L. Hennessy and David A. Patterson, 6th edition
2. Assignment Solutions