Problem 8.1

Let H be the hit rate of the L2 cache and A be the accuracy of the L2-cache hit/miss predictor. The following table gives the penalty associated with the four scenarios.

Table 67: Penalties

Hit/Miss	Prediction	Outcome	Fraction	Penalty
Hit	Hit	Success	H*A	0
Hit	Miss	Failure	H*(1-A)	120
Miss	Hit	Failure	(1-H)*(1-A)	60
Miss	Miss	Success	(1-H)*A	20

Therefore the total penalty with the added predictor is:
H*(1-A)*120+(1-H)*(1-A)*60+(1-H)*A*20
The penalty without any predictor is: (1-H)*60.
The predictor is beneficial if (1-H)*60>H*(1-A)*120+(1-H)*(1-A)*60+(1-H)*A*20
or 60-60H>60H-40A-80H*A+60
or A>3H/(2*H+1)
If H=0.50 the predictor is beneficial if A is at least 75%.
If H=0.80 the predictor is beneficial if A is at least 92.3%.

GPGPU Problem

A CUDA kernel A is launched with 10 blocks of 1024 threads on a GPU with 10 streaming multiprocessors (SMs) taking 27 seconds (s) to run. In this exercise we consider several scenarios in which the kernel is launched on a GPU that has 9 SMs but is otherwise equivalent to the first GPU. In both GPUs the maximum number of threads per SM and the maximum number of threads per block is 1024. Each SM can execute up to two warps in parallel. Assume also that the CUDA kernel does not make use of any `__shared__` memory.

(a) Suppose that kernel A is launched again with 10 blocks but this time on the GPU with 9 SMs. What is the execution time for kernel A on the 9-SM GPU?

(b) How long does it take for kernel A to run on the 9-SM GPU if launched with 9 blocks of 1024 threads, but doing the same amount of work as the 10-block launch.

(c) A different CUDA kernel B is launched with 10 blocks of 32 threads taking 72s on the 10-SM GPU. How long would the 10-block launch take on the 9-SM GPU?

(d) Given the following CUDA kernel code. Each instruction (A to E) costs 10 cycles. The warp size in the 10-SM machine is set to 32. The foo and bar vector sizes are 2048.

```
__global__ void ComputeW (float *foo, float *bar) {
  float v, w;
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
A: v = foo[tid];
B: if(tid < blockDim.x/2)
C:      v++;
   else
D:           v--;
E: w = bar[tid] + v;
}
```

Assume that the programmer set <<<64, 32>>>, which means 64 blocks and the block size is 32, what will be the execution time of the kernel?
If the configuration is changed to <<<32, 64>>>, what will be the new execution time?

Solutions

Solution to (a): In this case, because of the thread limit, only one block at a time can run on an SM. When the kernel is launched each of the first 9 blocks will be sent to an SM. The tenth block will have to wait for one of the first nine blocks to finish. Assuming that all blocks take about the same amount of time, that should occur in 27 s. The tenth block will then finish 27 s later, for a total time of 54 s.

Solution to (b): Since the number of blocks matches the number of SMs, we would expect that the execution time would be proportional to the amount of work divided by the amount of computing resources. If the amount of computing resources (the number of SMs) changes from 10 to 9, we would expect the execution time to increase by a factor of 10/9 yielding a run time of (10/9) * 27s = 30s.

Solution to (c): It would still take 72 s because with 32-thread blocks it's possible for two blocks to run simultaneously on the same SM. Since 32 threads are not nearly enough to fully utilize an SM's resources, the two blocks can run at the same time and they will take the same amount of time compared with that when each block was running on a separate SM, assuming that the code was not data-limited.

Solution to (d): For <<<64, 32>>>, due to the thread divergence in a warp, the execution time is 10(A)+10(B)+10(C)+10(D)+10(E) = 50 cycles.
However for <<<32, 64>>>, the issue is solved, the execution time is 10(A)+10(B)+10(C or D)+10(E) = 40 cycles.

Problem 4

Consider a future 8-way CMP on which you can power off some cores to allow the rest to operate at a higher frequency. You can use either 1, 2, 4, or 8 cores at the respective frequencies:

when only one core is running it operates at 0.3ns clock cycle
when two cores are running they operate at 0.4ns clock cycle
when 4 cores are running they operate at 0.5ns clock cycle,
when all 8 cores are running they operate at 1ns clock cycle,

Consider a partially parallel application which has a serial part that is 1000 Instructions and a parallel part that can be parallelized at will which is 2000 Instructions.
Parallelizing however requires you to create threads in the serial part of the application and 100 Instructions are added for every thread you create.
The serial and parallel part of the applications all run one instruction per cycle regardless of the frequency.

(a) Calculate the runtime of the application for the above processor when using 1,2,4,8 cores at their respective frequency without reconfiguring the processor while the application is running.
Which configuration is better?

(b) What if you could reconfigure the processor to run the serial part with one core at 0.3 ns clock cycle and then configure it to 2 or 4 or 8 cores for the parallel part. What is the runtime for each case? Always consider that creating each thread takes 100 Instructions that are added to the serial part (and run on the single core at 0.3ns clock cycle).

Solution to problem 4:

(a)
1-Core: 1000 Instr serial + 2000 Instr parallel = 3000 Instr at 0.3 ns = 900ns
2-Core: 1000 Instr serial + (1000 Instr parallel) + 200 Instr to create 2 threads = 2200 Instr at 0.40 ns = 880ns
4-Core: 1000 Instr serial + (500 Instr parallel) + 400 Instr to create 4 threads = 1900 Instr at 0.5ns = 950ns
8-Core: 1000 Instr serial + (250 Instr parallel) + 800 Instr to create 8 threads = 2050 Instr at 1ns = 2050ns
→ 2 Core is the best configuration

(b)
Serial part is always 1000 Instr at 0.3ns = 300ns
The parallel part :
2-Core: 200 Instr to create the threads at 03.ns + (1000 Instr parallel at 0.4) = 60ns + 400ns = 460ns
4-Core: 400 Instr to create the threads at 03.ns + (500 Instr parallel at 0.5) = 120ns + 250ns = 370ns
8-Core: 800 Instr to create the threads at 03.ns + (250 Instr parallel at 1) = 240ns + 250ns = 490ns

So the total runtime for the reconfigurable processor is
2-Core: 300ns + 460ns = 710ns
4-Core: 300ns + 370ns = 670ns
8-Core: 300ns + 490ns = 790ns

Problem 6

A system architect has three choices for an on-chip interconnection network: a uni-directional ring (UR), a bi-directional ring (BR) and an NxN mesh network (NM). In addition, the architect has two choices for providing coherence between L1 caches: snoop-based (SB) or directory-based (DB). There are 16 cores on the chip. All cores have private L1 caches and they share a L2 cache that is divided in 16 banks, with one bank attached to each core. The latency to communicate between two cores connected directly by a link is 1 cycles. The router latency is one cycle per router that the packet goes through. For directory-based coherence the access time to the directory is 5 cycles. Assume that there is no contention in any link or the directory.

a. Which combination of design choices provides the shortest latency to provide coherence? In this context, latency is considered to be the time it takes to reach all potential destination cores (not including acknowledgments). Consider the worst and average case for a coherence message to reach its destination(s)

b. Now consider 256 cores and a directory access time of 15 cycles, which combination of design choices provides the shortest latency to provide coherence in this case?

Please organize the results in a table as follows:

Case (a) 16 cores

Latency	**Coherence Type**	**UR**	**BR**	**NM**
Worst Case	SB	15	8	6
Worst Case	DB	5+15	5+8	5+6
Average	SB	15	8	5
Average	DB	5+8	5+4	5+3

Case (b) 256 cores

Latency	**Coherence Type**	**UR**	**BR**	**NM**
Worst Case	SB	255	128	30
Worst Case	DB	15+255	15+128	15+30
Average	SB	255	128	23
Average	DB	15+8	15+4	15+15