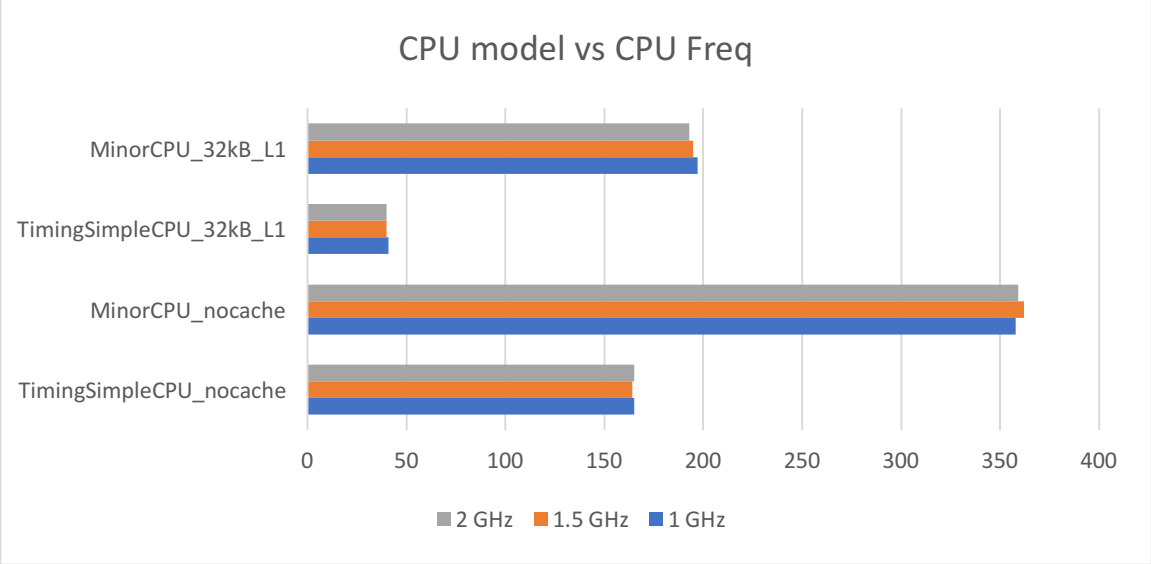
Write and submit a file named **report.pdf** containing a short report with your observations and conclusions from the experiment. This report should contain answers to the following questions:

- Which CPU model is more sensitive to changing the CPU frequency? Why do you think this is?
  - ⇨ The following table shows the time taken along with change in frequency for non-cached system and cached, both with DDR3_1600 RAM.

	Time for Freq (in seconds)				
	1 GHz	1.5 GHz	2 GHz	2.5 GHz	3 GHz
TimingSimpleCPU_nocache	165	164	165	165	167
MinorCPU_nocache	358	362	359	355	356
TimingSimpleCPU_32kB_L1	41	40	40	40	40
MinorCPU_32kB_L1	197	195	193	196	193



Based on the chart we obtain from the table above, we see that **MinorCPU** is more sensitive to the CPU frequency.

According to the tutorial, TimingSimpleCPU

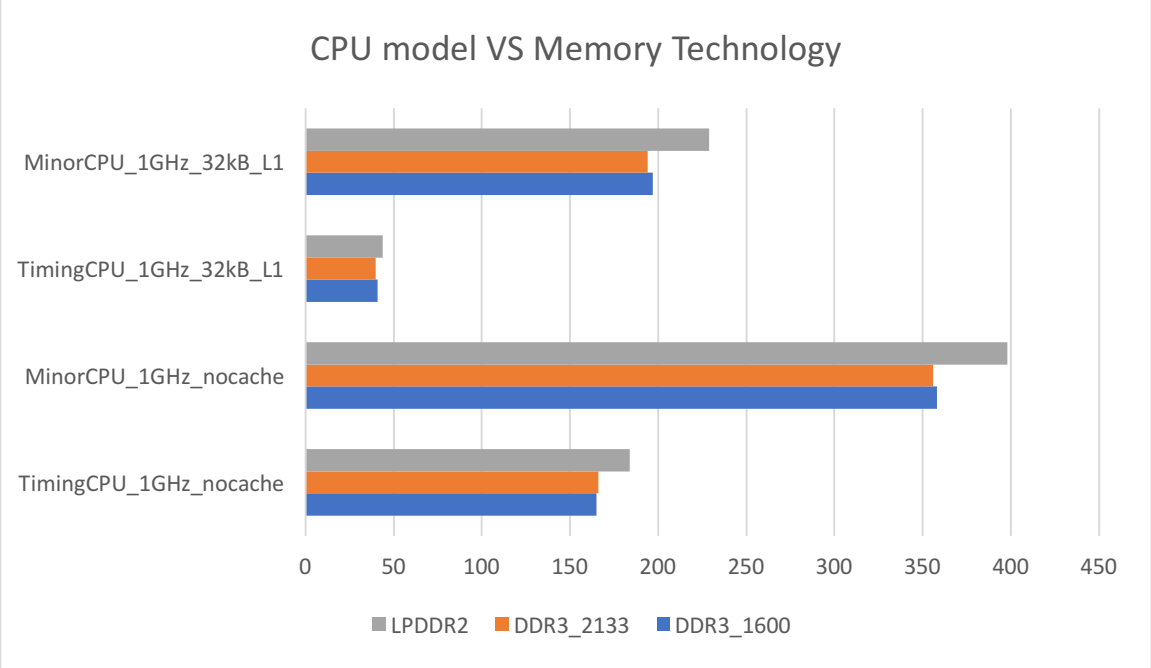
> "executes each instruction in a single clock cycle to execute, except memory requ ests, which flow through the memory system."

Which means, for a computational intensive program running in TimingSimpleCPU, increasing CPU frequency will lead to faster execution. However, our sieve benchmark is a memory intensive program. So, it will not lead to too much faster execution for TimingSimpleCPU.

However, for MinorCPU, it is a pipeline based CPU model. This means that after the pipeline fills up, it is most likely to have one instruction processed per cycle. So, increasing the CPU frequency would lead to a faster instruction processing.

- Which CPU model is more sensitive to the memory technology? Why?

	DDR3_1600	DDR3_2133	LPDDR2
TimingCPU_1GHz_nocache	165	166	184
MinorCPU_1GHz_nocache	358	356	398
TimingCPU_1GHz_32kB_L1	41	40	44
MinorCPU_1GHz_32kB_L1	197	194	229



Here, for TimingCPU_1GHz_nocache: change = (184 – 165) / 165 = 0.115
And for MinorCPU_1GHz_nocache: change = (398 – 358) / 358 = 0.112

Based on the chart we obtain from the table above, we see that **TimingSimpleCPU** is more sensitive to the CPU frequency.

I think it is so because memory requests in TimingSimpleCPU model flows through the memory system, which means that it is dependent on the memory technology. And for our sieve benchmark, since it is memory intensive, we can see the sensitivity more too as opposed to the MinorCPU model where once the pipeline is full, it has higher chances to have one instruction per cycle.

- Is the sieve application more sensitive to the CPU frequency or the memory technology? Why?
  - ⇨ The sieve application is more sensitive to the memory technology because we are accessing memory far too many times than we perform complex computation.

    In our case, let N be such that we are trying to find the number of prime numbers under N, M is the number of memory accesses, then,

    - We initially instantiate N memory location, so M = 1
    - Then we add 1 as default value for all location, so M = M + N = N + 1
    - Then for sqrt(N) times, we change the value in location containing multiple of each prime number to 0, so M = M + (sqrt(N) * N/2) for worst case, so M = N + 1 + N * sqrt(N)
    - We perform one square root calculation, so complex calculation, C = N * sqrt(N)
    - Then we count the number of 1s in memory, so M = M + N = 2N + N * sqrt(N) + 1

      In total, we have about N + 1 + N * sqrt(N) memory accesses and N * sqrt(N) computation. The computation can be reduced to N/2 + 1 if we precompute the value of sqrt(N). But, the memory accesses can only be reduced to N + N * sqrt(N) + 1 which is more than N/2 +1.

      Hence, the sieve application is more sensitive to memory technology.

- Why does the smaller direct-mapped L1 cache perform better/worse/similar to the larger direct-mapped L1 cache? Why does it perform better/worse/similar to the same-sized 2-way L1 cache? Why does the larger direct-mapped L1 cache perform better/worse/similar to the smaller 2-way L1 cache?
  - ⇨ The following are when mem technology is LPDDR2 and for MinorCPU

	Time/s	
Cache	1 GHz	3 GHz
32kB L1 direct-mapped	229	226
64kB L1 direct-mapped	220	220
32kB L1 2-way associated	216	195

  - ⇨ The smaller direct-mapped L1 cache performs **worse** than larger direct-mapped L1 cache because having a larger L1 cache would decrease the miss rate as L1 will be able to store more and there will be less chance for conflict, thereby reducing AMAT.

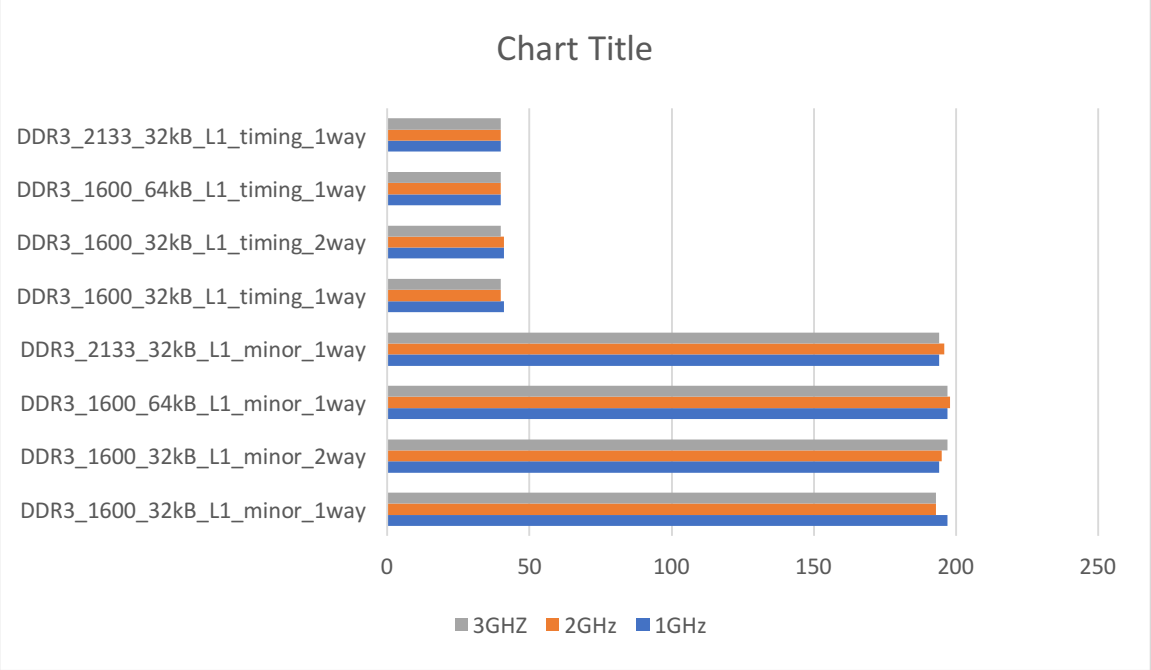
  - ⇨ The smaller direct-mapped L1 cache performs **worse** than the same-sized 2-way L1 cache because caches with higher associativity means less chance of a conflict between two addresses. This in turn reduces AMAT and memory stall cycle.

  - ⇨ The Larger direct-mapped L1 cache should perform **similar** as the smaller-sized 2-way L1 cache because higher associative cache means less conflict and so does having double the L1 cache size, especially when memory access is intensive but always in sequential way.

- **Suppose it costs the same with respect to the cache described in step #3 above and the DDR3_1600_x64 memory to either: increase the associativity of the L1 cache**

from 1 to 2, double the size of the L1 cache, or use the DDR3_2133_x64 memory. Which of these changes, if any, should you choose? Why?

	Time/s		
	1GHz	2GHz	3GHZ
DDR3_1600_32kB_L1_minor_1way	197	193	193
DDR3_1600_32kB_L1_minor_2way	194	195	197
DDR3_1600_64kB_L1_minor_1way	197	198	197
DDR3_2133_32kB_L1_minor_1way	194	196	194
DDR3_1600_32kB_L1_timing_1way	41	40	40
DDR3_1600_32kB_L1_timing_2way	41	41	40
DDR3_1600_64kB_L1_timing_1way	40	40	40
DDR3_2133_32kB_L1_timing_1way	40	40	40



⇨ If it cost the same then I would choose to use **DDR3_2133_x64** because
- It has similar performance advantage as a two-way associative cache
- It will be more efficient when we have a TimingSimpleCPU and more computational intensive programs running

- **If you were to use a different application, do you think your conclusions would change? Why?**

- ⇨ Yes, if I was using a different application the conclusion would change because the new application could have
  - More randomness in the way it accesses memory, which would then give us completely different value for each cache types
  - Less memory block needed for executing the program, which would then allow all types of caches to give great performance
  - Use an array instead of heap memory, which is more sequential