As it can be seen by the UML diagram that I created. We see that the CPU has an aggregation relationship with the computer of multiplicity 1 to 1 through the pointer ATTACHED\_CPU. This means that each part has its own lifecycle however ownership exists here. A computer object can be created without a CPU but isn’t technically functional unless it has a CPU. A CPU however is functional by itself, it doesn’t need a Computer to exist. The owner in this aggregation relationship is the computer as it has a pointer to a CPU object, while a CPU doesn’t. This means that its also unidirectional relationship. Also regarding the multiplicity, it’s a 1 to 1 relationship. A Computer can have a single CPU attached to it at one time, also, eg. if the CPU is being used by computer “A”, computer “B” can’t use the same CPU object. Regarding visibility here, the computer class, if it creates an object and it gets a CPU attached to it, that object can access the CPU’s execute function and constructor as they are public functions, this means there is an attribute visibility for the class, CPU is an attribute of Computer “here it is an attribute through a pointer”. Moreover, there exists parameter visibility through the operator+ overloaded functions and the setters for the CPU and GPU attachments. Those functions take a reference to the respective object as a parameter, this means that they can access those objects within the class, which is utilized to set the pointer attributes to be used later with the execute functions. The analogy and explanation that I used for the CPU and the computer is the same as for the GPU and the computer. I will not write it again is its straight up the same thing with GPU used instead of CPU and ATTACHED\_GPU instead of ATTACHED\_CPU.

Regarding the relationship between the ALU and the CPU, it’s a composition relationship with multiplicity 1 to 1 through the ALU object created in the CPU class with name ALU\_OBJ. Here we can say that the ALU belongs to the CPU, or it’s a part of the CPU. In our design, we gave the initialization job of the NUM\_PER\_CORES attribute of the ALU to the CPU actually, as specified in the question, then its delegated to finish in the ALU. This makes the ALU dependent on the existence of the CPU thus it’s a composition. Also a CPU can only have one ALU object and the ALU as per logic can be used by only one CPU that explains why it is a 1 to 1 multiplicity. Regarding visibility, the CPU object can set the private attribute NUM\_PER\_CORES of the ALU\_OBJ it has, it can also access all its public functions, that means it has as an attribute visibility and local visibility as its an object defined as an attribute “not taken from a parameter”, ALU is an attribute of CPU ”attribute through the ALU\_OBJ object”. Again, same explanation is valid for the relationship between the GPU and the CUDA classes. I will not write it again as its straight up the same thing, GPU = CPU, CUDA = ALU, NUM\_CORES = NUM\_PER\_CORES, CUDA\_OBJ = ALU\_OBJ and so on. The relationship is also unidirectional as can be seen per the UML diagram.

Generally speaking the execute function in all of the classes it was implemented in can be considered to have a parameter visibility but I don’t think mentioning it was important since even though it’s a class we usually act as if it’s a predefined data type.