Problem 1

Describe Roofline analysis. Describe/analyse Arithmetic intensity in the context of sparse matrix(in CSR - compressed sparse row format) multiplication vs dense matrix multiplication.

The most basic Roofline model can be visualized by plotting floating-point performance as a function of machine peak bandwidth, and arithmetic intensity. The resultant curve is effectively a performance bound under which kernel or application performance exists.

The arithmetic intensity (I), also referred to as operational intensity, is the ratio of the Floating point operation(F) to the memory traffic/Bytes utilized (B)

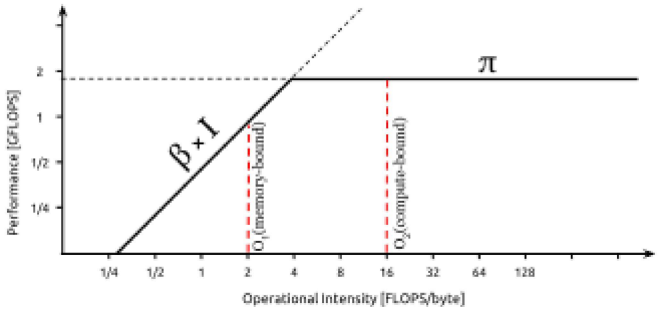
The memory traffic denotes the number of bytes of memory transfers incurred during the execution of the kernel or application. In contrast to Floating operation(F), B is heavily dependent on the properties of the chosen platform, such as for instance the structure of the cache hierarchy.

Note that the F is a property of the given kernel or application and thus depend just partially on the platform characteristics.

The naïve Roofline is obtained by applying simple bound and bottleneck analysis. In this formulation of the Roofline model, there are only two parameters, the peak performance and the peak bandwidth of the specific architecture, and one variable, the arithmetic intensity.

$$P = \min \begin{cases} \pi \\ \beta \times I \end{cases}$$

where P is the *attainable performance*, π is the *peak performance*, β is the *peak bandwidth* and I is the *arithmetic intensity*.



A sparse matrix is a matrix with enough zeros that it is worth taking advantage of them

Compressed column format is used to store optimally sparse matrix.

CSR representation will have higher arithmetic intensity as number of bytes utilized is reduced and thus in comparison to dense matrix, Arithmetic intensity of sparse matrix will be higher than dense matrix.

From Roofline analysis, performance of csr will be near to the peak performance as compared to dense matrix.

Problem 2

Demonstrate Inheritance by taking your favorite example in your favorite programming language supporting OOP.

Inheritance is a process that allows a class to reuse member variables and methods of another class. The class that inherits member variables and methods from another class is called a derived class or child class and the class that is being inherited by a child class is called a base class or a parent class.

Let's define a class Cloth, with child classes Jeans, Shirt and Skirt:

```
class Cloth:
    def cloth_details(self):
        self.name = "cloth"
        self.category = "category"

    def show_cloth_details(self):
        print("The name of parent class is ", self.name)
```

The Cloth class contains two instance variables name, category, defined inside an instance method cloth_details().

The Cloth class also contains an instance method show_cloth_details() which prints the name of the cloth.

Next, we define Jeans, Shirt and Skirt classes that inherit the cloth class.

```
class Jeans(Cloth):
    def jeans_details(self):
        self.waist_size = 34
        print("A Jeans has", self.waist_size , "waist size")

class Shirt(Cloth):

    def shirt_details(self):
        self.shirt_texture = "box texture"
        print("A shirt has", self.shirt_texture)

class Skirt(Cloth):
    def skirt_details(self):
        self.skirt_length = 20
        print("A skirt has", self.skirt_length, "skirt length")
```

The Jeans class has its own method jeans_details() and a member variable waist size, which means that in addition to having access to the name, category member variables of the parent Cloth class, the Jeans class has its own set of member variables.

Similarly, the Shirt and Skirt classes have the methods shirt_details() and skirt_details(), respectively. The shirt_details() method prints the value of the shirt texture variable of the shirt class whereas the skirt_details() method prints the value of the skirt length.

The basic idea of inheritance is that the methods and variables that are common in multiple child classes are included in the parent class. The methods and variables that are specific and not common among child classes are added in respective child classes.

For example, since not all cloth has waist size, therefore the variable waist size has not been added to the parent Cloth class, rather it is added to the child Jeans class. Similarly, the variable shirt texture and skirt length are intrinsic to respectively shirt and skirt classes, therefore these variables are added in the respective child classes and not in the Parent Cloth class.

The following script creates objects of the Jeans, shirt and skirt classes and access the parent and child class methods and variables:

```
jeans = Jeans()
jeans.cloth_details()
jeans.show_cloth_details()
print(jeans.category)
jeans.jeans_details()
```

✓ 0.3s

```
The name of parent class is  cloth
category
A Jeans has 34 waist size
```

Problem 3

1. **What is 'public', 'private' and 'protected' in C++? In line number 17, why is the word 'public' used?**

   C++ offers the possibility to control access to class members and functions by using access specifiers. Access specifiers are used to protect data from misuse.

   ### Public Specifier

   Public class members and functions can be used from outside of a class by any function or other classes. You can access public data members or function directly by using dot operator (.) or (arrow operator-> with pointers).

   ### Protected Specifier

   Protected class members and functions can be used inside its class. Protected members and functions cannot be accessed from other classes directly. Protected data members and functions can be used by the class derived from this class.

Private Specifier

Private class members and functions can be used only inside of class.

Why do derived class inherit public base class?

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance.

When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

In our case, show and print member functions of base class are inherited to derived class and can be accessed from object of derived class from main program or polymorphism is possible.

2. **What is a virtual function? Why is function 'print' declared as virtual in line number 6, but not in line number 19?**

   A C++ virtual function is a member function in the base class that you redefine in a derived class. It is declared using the virtual keyword. It is used to tell the compiler to perform dynamic linkage or late binding on the function. Although C++ permits the base pointer to point to any object derived from the base class, it cannot directly access the members of the derived class. Therefore, there is a need for virtual function which allows the base pointer to access the members of the derived class.

   Virtual keywords will be used in the base class and derived class will just override in their class. Thus, line number 16 is declared as virtual but not in line number 19.

3. **What is inheritance? Explain with respect to the above code.**

Inheritance is the ability of one class to inherit another class. Inheritance provides reusability of code. Here base is a parent class and derived is a child class. Child class inherits base class and thus, can inherit base class attributes, methods, and properties. Here when we call the show method from the child class, compiler invoked the inherited method.

4. **What is polymorphism? Does the above code contain an instance of polymorphism?**

   Polymorphism means multiple forms. We can find the same function taking multiple forms with different number of attributes. In the above code we can see one type of polymorphism which is function overloading. Here same function with name "print", one having zero argument and other argument are defined. Here d.print(1) will call the derived class print function with one argument and print the number while d.print() will call the derived class print function with zero argument and print "print derived class".

5. **What will be the output of the code?**

   Output:

print base class

show base class

print derived class

show base class