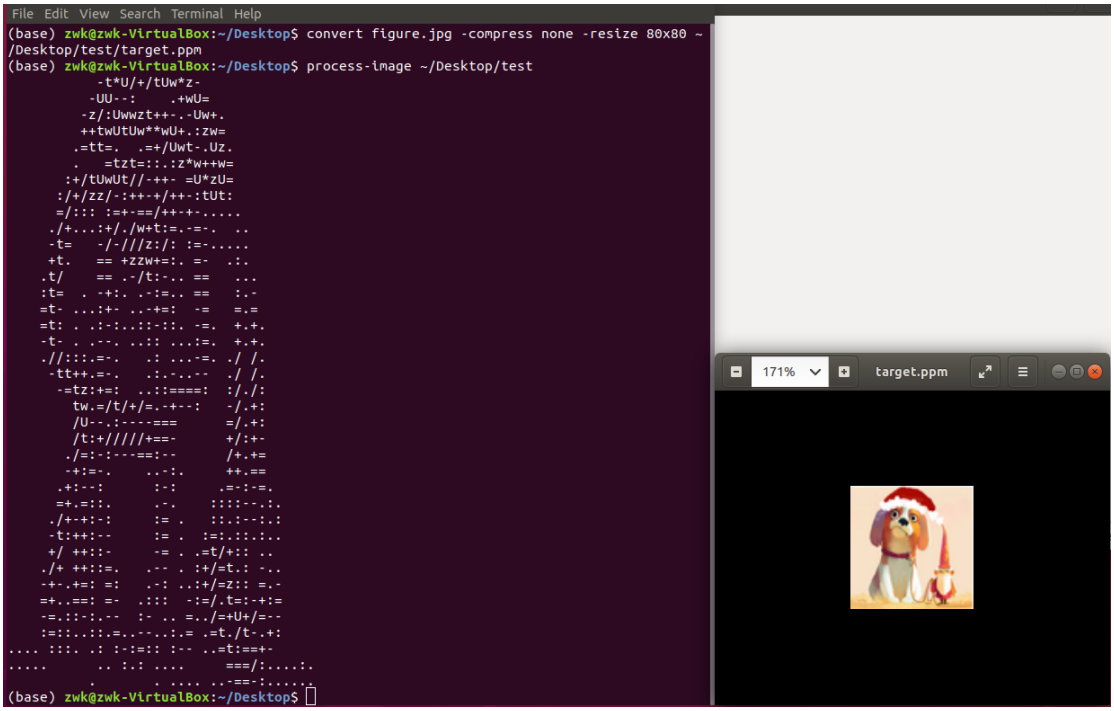
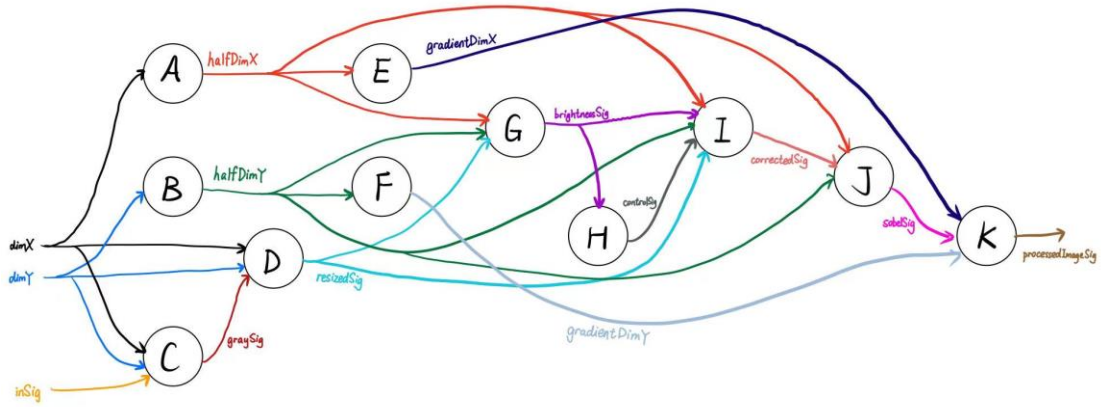
5 (b)



5 (c)



A: halfDimXActor
B: halfDimYActor
C: graySDF
D: resizeSDF
E: gradientDimXActor
F: gradientDimYActor
G: brightnessSDF
H: controlSDF
I: correctionSDF
J: sobelSDF
K: asciiSDF

5 (d)

A: halfDimXActor, divides the input signal dimX by 2
B: halfDimYActor, divides the input signal dimY by 2
C: graySDF, transforms every RGB pixel into a grayscale one
D: resizeSDF, the X and Y dimension of the image get halved so that the total size of the image is a quarter of the original size
E: gradientDimXActor, subtracts 2 from the signal halfDimX

F: gradientDimYActor, subtracts 2 from the signal halfDimY
G: brightnessSDF, outputs the maximum and minimum bright pixels for every image in the stream
H: controlSDF, keeps an average of levels for some images in the stream and outputs the control signal for the latest image being processed
I: correctionSDF, outputs a brightness correction according to its function
J: sobelSDF, applies the Sobel operator in the image stream to detect edge
K: asciiSDF, outputs the ASCII "art" of the image stream

Overall, the SDF converts a colorful image into a stream of ASCII characters, which is output in the terminal and looks like the shape of the original image. It first converts image pixels into grayscale equivalent and then resize the image. According to the resized stream, it gets the maximum and minimum bright pixels of the picture and rescale the resized stream based on the maximum and minimum bright pixels. After that, it applies sobel operator in the stream and use the sobeled signal to generate the ASCII stream of characters to represent the original image.

5 (e)
dimX: Int, 32 bits
dimY: Int, 32 bits
inSig: Signal Int, 3 * X * Y * 32 bits
halfDimX: Int, 32 bits
halfDimY: Int, 32 bits
gradientDimX: Int, 32 bits
gradientDimY: Int, 32 bits
graySig: Signal Double, X * Y * 64 bits
resizedSig: Signal Double, (X / 2) * (Y / 2) * 64 bits
brightnessSig: Signal Double, 2 * 64 bits
controlSig: Signal Control, the size of the token is hard to say because we need to know how this signal is implemented by the compiler. For example, if it uses Int to label the control signal, the size will be 32 bits.
correctedSig: Signal Double, (X / 2) * (Y / 2) * 64 bits
sobelSig: Signal Double, (X / 2 – 2) * (Y / 2 – 2) * 64 bits
processedImageSig: Signal Char, (X / 2 – 2) * (Y / 2 – 2) * 8 bits (if a char is 8 bit)

5 (f)
If the actors A, B, C, …, K have runtime a, b, c, …, k respectively, we can get the run-time of the application running on a single processor as a + b + c + … + k.

5 (g)
If we assume every operation takes the same time, then we can get
A: halfDimXActor: 1
B: halfDimYActor: 1
C: graySDF: X * Y
D: resizeSDF: (X / 2) * (Y / 2)
E: gradientDimXActor: 1

F: gradientDimYActor: 1
G: brightnessSDF: (X / 2) * (Y / 2) * 2
H: controlSDF: 2
I: correctionSDF: (X / 2) * (Y / 2)
J: sobelSDF: This one is complicated. It involves convolution, square and square root. By rough estimation, it should have the longest run-time compared to others.
K: asciiSDF: (X / 2 – 2) * (Y / 2 – 2)
Therefore, sobelSDF > graySDF > brightnessSDF > resizeSDF ≈ correnctionSDF > asciiSDF > controlSDF > halfDimXActor ≈ halfDimYActor ≈ gradientDimXActor ≈ gradienDimYActor

5 (h)
For the main actors (graySDF, resizeSDF, brightnessSDF, correctionSDF, sobelSDF and asciiSDF), there are strong data dependencies that they rely on the signal produced by the immediate predecessor. Such situation can make parallel implementation useless because they need to wait for the predecessor to finish and generate the signal. Therefore, the time will not be saved. However, we can use pipelined structure to improve. For example, the execution process of one actor is divided into several stages and only one stage will use the signal produced by the predecessor. When the signal is not produced yet, we can start the stages that will not use the signal. When the signal is produced, we can store it in a register file or buffer and the stages that will use the signal can have access to it. Through pipelined structure, we can know handle the data dependencies and adopt parallel implementation.