**Part C.3**

Our model can run in bounded memory provided, each behavior runs at similar rate, if not then, it cannot run in bounded memory. For e.g, stimulus runs at a very fast rate but susan runs very slow, then as time progress, the size of queue grows larger and larger (no bound) . The model can execute with queue size = image size. Limiting the queue size affects the scheduling by bring an order (corresponding to data flow) to execution. Execution proceeds as  
Stimulus → read\_image → susan → write\_image → monitor   
  
Even though read\_image , susan, write\_image are parallel behaviors , they complete sequentially because of the data dependencies. Susan and write\_image will start executing but will wait on queue.receive for data from its preceding behavior. Also read\_image cannot execute again until susan gets (consumes the data in queue) once.

Having queue size greater than image size, decouples the behavior from its subsequent behaviors, for e.g Stimulus can send images continuously to read\_image (till queue becomes completely filled) .

As long as all the behaviors are working as they are meant to be, and queue size is at least 1 image size , the model is deadlock free and it is deterministic, as there is no shared resource.

**Part D.2**

We chose to parallelize edge draw behavior. In our implementation , we are instantiating 2 instances of edge\_draw behavior which processes half image parallely. Edge\_draw requires 2 inputs – mid and input\_image. We created separate behavior for splitting the mid and image into 2 sections which can be sent each of the edge\_draw instance. Within edge\_draw, processing each image element modifies few image elements before and after the specific element in consideration. We split the image into 2 parts with overlapping sections, the image is 7220 (IMG\_SIZE) elements long, 1 part is image [0: (IMG\_SIZE/2 + offset)] and 2nd part is image [(IMG\_SIZE/2 – offset) : IMG\_SIZE-1], where offset is 133 (decided based on input and dependencies. The processed outputs from each edge\_draw instance is then merged together using another newly created behavior.

# Add behavior hierarchy sir\_tree -blt command

# graphical representation of model (screenshot from specC viewer)

This model implemented as KPN shows the task level parallelism but the the parallelism within a task is not explicitly highlighted. There is potentially parallelism to be exploited in susan\_thin and susan\_Edge behavior which is unknown until implemented parallely.   
Implementing it using different MOC (like Data flow graph with finer granularity) would show the parallelism within task more explicitly.

Yes, this model can be modeled using SDF, . For SDF implementation , we will have to somehow initialize the inputs for all the parallel behaviors so that they don't wait on others for their output during the initial run.

#could a tool be developed to recognize if a model is KPN or SDF ?  
 - if feedback , KPN fails (might be wrong ?)  
 - if can statically schedule it is an SDF model ??  
 -

#Adv and Disadv for KPN vs SDF