Q1: Is OpenACC supported on Azure/AWS?
A: This mainly depends on the availability of an OpenACC compiler supporting the available hardware. E.g. the qwiklabs environment we use for the homework use AWS cloud instances which have the PGI compiler installed and have NVIDIA GPUs. One could install an OpenACC compiler within an AWS or Azure instance, but may have to generate a new license file each time a new instance is used.

Q2: On slide 44, on the chart, N= represents the number of processors or multicore GPU.
A: It represents the number of single threaded MPI ranks and thus correspond to CPU cores.

Q3: Do all of these examples assume that all the data can fit on the GPU simultaneously?
A: Yes the presented examples assume that the full problem fit into the aggregated GPU memory.

Q4: Could you please elaborate on why MPS is "for legacy MPI applications"?
A: Because a legacy MPI application might have a very large code base and not all routines have been accelerated with OpenACC yet and thus although good GPU speedup for some routines is achieved its required to also use all CPU cores with MPI to achieve an overall application speedup. This is summarized on slide 44.

Q5: With PGI Fortran compiler can we use mpirun, mpicc etc.?
A: You can use MPI with PGI Fortran as with C. Just use the MPI compiler wrapper for Fortran mpif90.

Q6: Is it straightforward to stream data to GPU with OpenACC?
A: Yes this can be done with the async clause. This topic was covered in "Class #4 - Advanced OpenACC Techniques: Interoperability, MPI, and Pipelining" of the OpenACC Overview Course: https://developer.nvidia.com/openacc-overview-course

Q7: Does streaming work with MPS?
A: Yes using the async clause also works with MPS. But you need to consider that multiple processes are sharing the resources of a single GPU.

Q8: How will this operations run on a GTX 960 (4GB DDR5) and GTX 730 (1Gb DDR5) on i7 on Linux/Windows?
A: Yes this should work on the described hardware, but load balancing issues need to be resolved to get a speedup when paring the GTX 960 and GTX 730.

Q9: Does slide 92 refer to PCIe 2.0 or 3.0?
A: The K40 is connected with PCIe 3.0 the inter GPU bandwidth is limited by the infiniband network.

Q10: Does CUDA.aware MPI uses GPUDirect automatically?
A: All CUDA-aware MPI implementations use GPUDirect P2P is used automatically if available. Depending on the used MPI implementation GPUDirect RDMA needs to be enabled. E.g. for

OpenMPI GPUDirect RDMA can be enabled with --mca btl_openib_want_cuda_gdr 1: http://www.open-mpi.de/faq/?category=runcuda#mpi-cuda-support.

Q11: Can we use MPI in PGI Fortran along with OpenACC?
A: Yes, the same technique can be used to mix OpenACC and MPI in Fortran. We have an example of this available at https://nvidia.qwiklab.com/.

Q12: I'm familiar with MPI for multiple processor and threading for multicore processor. What are your thoughts?
A: Programming MPI+OpenACC is very similar to MPI+OpenMP. E.g. with MPI+OpenMP you would start one MPI rank per CPU socket and run as many OpenMP threads as there are cores in each socket. With MPI+OpenACC one usually starts one MPI rank per GPU using OpenACC to expose the parallelism for the GPU.

Q13: If each MPI node is multiple cores and a single GPU, should we use OpenMP for the cores and OpenACC for the GPU?
A: Some programs will run best with MPI between nodes, OpenMP on CPU cores, and OpenACC on the GPU. Others choose to run multiple MPI ranks per GPU using the multi process server (MPS). Using multiple MPI ranks is often simpler, unless you already use OpenMP.

Q14: Can we use MPI send receive inside compute regions?
A: No that is not possible. MPI is host only API so the MPI communication needs to be coordinated by the CPU. However with a CUDA-aware MPI references to GPU memory can be directly passed to MPI and if GPUDirect P2P or RDMA is available data is copied directly from GPU to GPU without touching the CPUs memory.

Q15: If all GPUs are installed on one computer, do we need to use MPI?
A: No you don't need to use MPI. There two main alternatives: Either control all GPUs from a single CPU thread by switching between them with acc_set_device_num or use a dedicated CPU thread for each GPU, e.g. combining OpenMP and OpenACC. Compared to these approaches MPI has two advantages: 1. It allows to scale beyond a node. 2. It allows to utilize GPUDirect P2P for direct GPU to GPU communication.

Q16: Would it be inappropriate to try to use OpenACC for both the multiple cores and the GPU?
A: No that would not be inappropriate, but currently it would require you to do some tricks to make it work, because OpenACC does not provide a way to divide your loop iterations between the GPU and CPU automatically. E.g. you could start one MPI rank per GPU and CPU and let the MPI ranks using the CPU use the multi core target. With the PGI compiler this currently this requires to start different binaries for the CPUs and the GPUs because it's not possible to build a fat binary containing GPU and Multicore CPU codes, although this is a temporary limitation. The main complexity with this approach balancing the load between the GPU and the CPU MPI ranks.

Q17: How do we know if we can use GPU-aware MPI distribution? Is there a run-time check we can do or do we need to know at compile-time?
A: Unfortunately there is no standard way to do this. OpenMPI offers a compile time and a runtime check for this: http://www.open-mpi.de/faq/?category=runcuda#mpi-cuda-aware-support

Q18: Can we use OpenACC for Xeon Phi Coprocessor?
A: There are not currently an OpenACC compilers for Xeon Phi, but PGI and others have announced planned support for KNL as they become more widely available.

Q19: There is an OpenMP4 compiler for Nvidia (Cray), will PGI and Intel converge and get along or will openmp4 and openacc continue to "compete"?
A: NVIDIA is working with Intel and others both within the OpenMP and OpenACC technical committees. We're committed to the future of both specifications. I'd suggest watching http://on-demand.gputechconf.com/gtc/2016/video/S6410.html for more information.