Q1: What does "unstructured" mean in the context of unstructured data movement?
A: Unstructured means that it is not tied to following scope afterwards, such that if you specify unstructured enter copying, and then when you exit the scope of the function the data will remain on device. A structured data region is limited to the scope in which it's used. For instance, the structured region must begin and end within the same function. An unstructured directive can span different scopes, including entering from one function and exiting from a function in another file.

Q2: Unstructured data directives, is it coming from OpenMP, or a new concept?
A: OpenMP 4.0 did not include unstructured directives, but OpenMP 4.5 will provide similar unstructured data directives to what OpenACC 2.0 has.

Q3: When we copy in the structure which contains embedded pointers that are pointing to host memory, OpenACC will automatically fix up those pointers to refer to the device copies?
A: No, you need to create memory on device for these pointers explicitly using create or OpenACC API and update corresponding data as necessary. Alternative option is to use Unified Memory which will handle this for you. The OpenACC committee is currently working on support for automating fixing the pointers in the 3.0 specification.

Q4: A gang would be similar to a block in CUDA, a worker to a thread?
A: A gang in OpenACC corresponds to a threadblock in CUDA. A worker in OpenACC conceptually maps to a warp in CUDA. However depending on vector length, a worker can consist of multiple warps, or multiple workers can span across a single warp. For example, vector_length(128) generates 4 warps per worker; vector_length(32) generates one warp per worker. You can also think of this as the following in CUDA terms: a vector maps to threadIdx.x, while a worker maps to threadIdx.y.

Q5: In which file can I see the code generated for the GPU after the code is compiled? Is necessary to ask for it using a compiler flag?
A: Yes, you need to specify keepptx flag to keep PTX assembly files after compilation, i.e. use -ta=telsa:keepptx. You can find generated PTX code in corresponding *.ptx files in the same folder. If you want to see internal CUDA assembly code you can later use cuobjdump utility on the binary, see http://docs.nvidia.com/cuda/cuda-binary-utilities/index.html for more details.

Q6: Are there plans to include multi-gpu support in OpenACC through directives without the need of using the API?
A: PGI currently has an extension to do this. It's not currently in the specification. We can raise this to the technical committee.

Q7: Can we do our experiments on any available remote system at NVIDIA or PGI using OpenACC?
A: Qwiklabs, GPU test drive, seeding program, public supercomputers (Titan, Bluewaters, Piz Daint).

Q8: Can you also explain headers that we may need to know about/use? I've found that I sometimes need to use accelmath.h to make things compile.
A: Usually no additional header files are required. OpenACC runtime library functions can be found in openacc.h in C/C++ or with “use openacc” in FORTRAN. The header file “accelmath.h” is provided by PGI for many common math routines, such as trig functions. If you are calling these types of math routines inside OpenACC compute regions, then accelmath.h will be required.

Q9: Data movement in OpenACC is Synchrounous or Asynchronous?
A: By default all data movement will be performed synchronously. In lecture 4 we will show an example of using the `async` clause to make the data movement asynchronous with the CPU and GPU.

Q10: Why not copying instead of create on v.coefs[:n]?
A: In this case, v.coefs is empty at the time we add the directive, so we use a `create` to avoid unnecessarily copying data from an uninitialized CPU array. In the matrix function the array is allocated and initialized in the same function, so we went ahead and used `copy` to do the creation and copy in one step.

Q11: Why is it required to free a variable in device memory before freeing it in the host memory?
A: OpenACC is designed as an offloading model, which assumes that execution and data begin on the CPU before being offloaded to the accelerator. This means that the CPU essentially owns the master copy of all arrays. If we free the CPU array first, the runtime will try to look-up the matching device array for an invalid host pointer, which may result in an error. By removing it from the GPU first, we ensure that we won’t encounter this error or leak the memory by removing the CPU reference first.

Q12: Have you compared the test case with CUDA? Which is more efficient?
A: We do not have a direct port of this test code to CUDA to perform this comparison. We’ve found that OpenACC codes where we are able to compare against a matching CUDA version typically achieves 80-90% of CUDA performance. It’s hard to make this comparison though, because it requires the code to have been written in both programming models and will depend greatly on the code and CUDA optimizations used.

Q13: Can the PGI compiler target the Intel Phi using the -ta:multicore option?
A: PGI 15.10 includes support for multicore X86 processors. It may be possible to run this code on a Phi as well, but it will not be well-optimized for Xeon Phi. PGI has announced a plan to fully support and optimize for Knights Landing in 2016, once it becomes commercially available.

Q14: how to link a pgc++ compiled code (blabla.a) with a main code compiled with c++ or g++ GNU compiler.
A: --gnu option in pgc++

Q15: With the kernels directive, is it possible to intersperse memory management calls between loops? I have code that does allocation in between loops inside of a Kernels directive. The code compiles fine, but I get an illegal instruction error and it's not clear to me whether this is workable? Something like #pragma acc kernels { for (...) {} ... malloc(...) ... for (...) {} }
A: No. You need to allocate data before kernels directives and use data directive to copy to be able to use in OpenACC loops within acc kernels.

Q16: Can the compiler target, then, Intel Phi?
A: PGI 15.10 includes support for multicore X86 processors. It may be possible to run this code on a Phi as well, but it will not be well-optimized for Xeon Phi. PGI has announced a plan to fully support and optimize for Knights Landing in 2016, once it becomes commercially available.

Q17: I understand that vector=thread and gang=block. I am still a little unclear what worker equates to?
A: Worker doesn't have a direct analogue in CUDA. From a CUDA programmer's perspective, one can roughly think of the vector length as the X dimension of a threadblock and the workers as the Y dimension. This is not strictly true, but is conceptually similar.

Q18: You can see the CUDA C kernels if you specify the -ta:tesla:nollvm option
A: This will dump the CUDA, but it's not very readable.

Q19: Can you give example of code using multiple GPUs?
A: There will be an example in next week's lecture.

Q20: So you want the reduction to be on a single gang?
A: Reductions can occur across gangs or within gangs, depending on where it's specified. The compiler will handle the complexity of reducing across gangs for you.

Q21: Is there a block-wise synchronization directive if one is using the acc parallel construct to have multiple loops within one kernel? (like synchthreads() in CUDA)
A: No, OpenACC places the burden of ensuring proper synchronization on the compiler, rather than the developer, so there's no directive for synchronization. In the case of multiple loops in one kernel, the compiler will handle the synchronization for you.

Q22: Can you show any example or using CUDA libraries within OpenACC?
A: This is a topic for next week's lecture. You can also find examples in https://github.com/NVIDIA-OpenACC-Course/openacc-interoperability.

Q23: In your experience, what is a good vector size for doing a reduction operator on a GPU?
A: It depends on the number of elements or loop iterations that you have in your reduction loop. For example if you have only a few iterations, you might want to specify the reduction and the vector size as 4 or 8. If you have 27, the good vector size will be 32. With larger sizes you might want to experiment with larger numbers.

Q24: So for a 3D block, would you use worker, worker, vector for the three loops?
A: I would use gang, worker and vector.

Q25: Any recursion can be recoded as an iteration. Iteration is faster than recursion because with iteration you don't keep putting arguments on the stack.
A: True, you rewrite your iterations using a recursion.

Q26: How are multidimensional arrays handled in C in OpenACC directives? Arrays of points or actual 2-D arrays?
A: Arrays of points you can handle them as arrays of points if you want to. For actual 2D arrays – don't have experience.

Q27: I assume it cannot handle a reduction loop followed by a loop that uses the result? Because that would require a GPU-wide synch within a kernel which cannot happen?

A: If you have 2 loops – 1st one is the reduction loop and the 2nd one uses the results of this reduction. If you parallelize them as 2 kernels, you can use the results because these loops will be executed as separate kernels. So you don't need any sync, because sync happens automatically in between the kernels, unless you specify an async close. If in one kernel – there is no guarantee that result will be updated.