OpenACC Course
Lecture #3 Q&A

Q1: Does OpenACC support x86 CPUs? What about Xeon Phi?
A: The OpenACC specification was designed to work on any parallel architecture, including both GPUs and CPUs. As of PGI 15.10, the PGI compiler officially supports building for multicore CPUs with the -ta=multicore target. This implementation has shown very good results so far. Support for Power CPUs and Xeon Phis are scheduled for 2016. Please see http://www.hpcwire.com/2015/10/29/pgi-accelerator-compilers-add-openacc-support-for-x86-multicore-cpus-2/ for more information.

Q2: can the source code be downloaded from somewhere?
A: The sources for the labs and lectures can be found at https://github.com/NVIDIA-OpenACC-Course/nvidia-openacc-course-sources.

Q3: What would be a good strategy for setting the size the number of workers for an x86 CPU?
A: We're still gaining experience using OpenACC on x86 CPUs. You will likely want a vector length that aligns with the width of the SIMD instructions on the CPU. It'd be worth experimenting with 1 gang per CPU core and 1 worker per hardware thread, but there may be other mappings that would make sense.

Q4: Does matrix multiply in OpenACC is speed up by explicit tiling?
A: Tiling is a common optimization for matrix multiplication and the code would likely perform better with tiling than without. However, when doing something common like a matrix multiplication, calling an accelerated library would perform significantly better. Interoperability with accelerated libraries will be discussed in the 4th lecture and examples can be found here: https://github.com/NVIDIA-OpenACC-Course/openacc-interoperability.

Q5: Couldn't you use vector and vector? Then it would create 2D blocks?
A: Some compilers will support doing this, but the officially supported way to achieve this type of decomposition since OpenACC 2.0 is to use the tile clause.

Q6: Do you know any PaaS / IaaS cloud providers that allow GPGPU programming (GPU access)?
A: The NVIDIA GPUs available on Amazon EC2 do support GPGPU programming. This is where the GPUs used by the qwiklab system are located, so performance should be similar to the performance achieved on the labs.

Q7: Does the compiler inform that it cannot determine the size of arrays?
A: Yes, it will print out the acceleration restriction message if you specify -Minfo an the compiler option.

Q8: Slide 12: unstructured Data, 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.

Q9: If one uses create() and then before the acc loops one wants to copy the data over, does one use the update clause?   On the loop?

A: Yes, you would use an “#pragma acc update” directive between the data region with the create() clause and the compute region (kernels or parallel loop) where it is used.

Q10: 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.

Q11: Is there a way to do conditional compilation at the preprocessor level? (e.g. #ifdef __ACC__)
A: Yes, there is a preprocessor macro _OPENACC which is defined when compiled with OpenACC directives.

Q12: 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 copyin, and then when you exit the scope of the function the data will remain on device. Structured data regions are always tied to the scope afterwards (i.e. loop or block statement { }). 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.

Q13: Is possible to see the code that the compiler generates for the GPU before it is compiled?
A: You need to compile the application to see the code generated for GPU.

Q14: Does that mean collective operations only run on 1 SM?
A: No, the collective operations (reduction, for example) can use all available SMs.

Q15: 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. 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.

Q16: A gang would be similar to a block in Cuda, a worker to a thread?
A: threadblock in CUDA = gang in OpenACC, worker in OpenACC can consist of multiple warps depending on vector length.

Q17: What do you mean by "portable"? Where else OpenACC code can run besides Nvidia card?
A: Additionally to NVIDIA GPUs, OpenACC runs on AMD GPUs, x86 and OpenPOWER CPUs, and on Xeon Phi in the near future. Additionally, OpenACC compilers are available from PGI, Cray, PathScale, GCC (limited support) and several university research groups.