OpenACC Course October 2018
Lecture 2 Q&As

Question	Response
Which Spec version we should follow 2.5 or 2.6?	2.6
What is the defined behavior of an "acc enter data copyin(a)" if "a" is an array that has not been allocated?	This will probably just seg fault at execution when the runtime attempts to copy data from an unallocated buffer.
Last week we saw that -ta=multicore triggers CPU computation wheres -ta=tesla:managed launches GPU computation. However, the loops were parallelized with OpenACC directives. HOw does the compiler change from OpenACC to OpenMP for CPU ?	The compiler doesn't change to using OpenMP, but I think you mean parallelizing accross multicore CPUs which is similar to OpenMP. With OpenACC you are defining which loops to parallelize. This parallelism is general so can be targeted to different architectures.
I am having GTX 1050 TI, then what flag will I be using?	GTX 1050 should be pascal architecture. That would be -ta=tesla:cc60 You can double check me though, if you are using PGI compiler there is a command called "pgaccelinfo" which will give you info (and compiler flags) for all accelerator devices on the system.
-ta=multicore triggers multi CPU computation, but how did the compiler do that if the instruction were OpenACC directives?	The directives are generic and the compiler knows how to translate the parallelism to different targets. The details are a bit much to explain in the forum.
How can I set a number of "threads" for a program with OpenACC directives, like we did with OpenMP on console (export OMP_NUM_THREADS) ?	When targeting multicore devices (-ta=multicore), you can set the environment variable "ACC_NUM_CORES" or, starting in PGI 18.10, call the API "void acc_set_num_cores(int);"
Can targeting GPU automatically handle multi-GPU parallelism? In other words, are the examples illustrated so far only for shared memory parallelism?	Using multiple GPUs requires additional work by the programmer. A single accelerator region will not automatically span multiple GPUs.
Followup: with PGI 18.4 it does not segfault (probably because I never try to use "a"). The use case is that I have conditional arrays that sometimes are used, sometimes not, all within the same structure. For simplicity, I have a single "data copyin()" call for all members, but sometimes some arrays in the struct are not allocated yet. It seems to work now, but I was wondering what the "official" behavior is according to the spec so that the code does not break with future updates/compilers.	You might want to look at using the "no_create" clause for the optional arrays, assuming that you're copying them individually and not the structure as a whole.
In FORTRAN, isn't the starting index and ending index optional if you want to copy the whole array?	Yes
Does OpenACC (PGI compiler) handles adding padding to 2D arrays necessary for correct align to the cacheline, for faster global GPU memory access?	No it does not. You may be thinking of CUDA pitched allocations. Pitched allocations are generally not used in OpenACC, and they are less advantageous on newer GPUs anyway.
Do the copy, copyin, copyout, ... also work on derived types in fortran if I have more complicated data structures?	In Fortran, if you have a derived type with allocatable data members, you can add the flag "-ta=tesla:deepcopy" to have the runtime perform a deep copy of the structure. The caveat being it will copy the entire structure including all sub-structures. So if you

Question	Response
	only want to copy a portion of the structure to the GPU, you'll want to stick to manual deep copy (unstructured data regions) or CUDA unified memory (-ta=tesla:managed). The next OpenACC standard also defines new directives for true deep copy ("shape" and "policy") which are prototyped in the PGI 18.10 compilers.
Did the webinar start? I cannot see anything.	The webinar has started. You may want to try refreshing your browser, make sure you have a good internet connection, make sure volume is turned up on your computer as well as the view window
I have two accelerator regions that are independent of each other in variables/arrays/memory and I would like each one to be parallelized on two different GPUs, would that be possible in OpenACC Fortran? So two independent loops, one goes to GPU1 and the other goes to GPU2. Is it possible for the host to send info to both and receive info from them async?	Yes, it's possible. You can find examples on the web demonstrating multi-GPU use in OpenACC. Here is one example: http://on-demand.gputechconf.com/gtc/2017/presentation/S7546-jeff-larkin-multi-gpu-programming-with-openacc.pdf
Q&A link from last week is broken	Here is the link to the Q&A document: https://www.openacc.org/sites/default/files/inline-files/OpenACC_Course_Oct2018/Lecture1_Q&A_2018.pdf
what is the latency (time wise) associated with a host device copy on modern GPUs? Must be milliseconds?	The latency consists of some overhead, plus a variable amount of time depending on the size of the copy. The overhead is usually 50 microseconds or less, and the variable duration for larger transfers can be computed by dividing the transfer size by the link bandwidth, which is often around 6 or 12 GB/s
No need to answer this if you think it is more suited for next lecture but: If I have a few "async(#)" kernels in a row, but one (or more) is in a conditional, I can currently use a simple "acc wait" and it will wait for whatever streams are being used. However, is there a way to be more selective about this. I.e. can I use a "wait(1,2,3,4,5) even if stream "2" let's say is not launched? The use case is that I want to have a more global async kernel that I do not want to wait for at the "wait" in question.	Yes, waiting on a stream that has had no instructions issued into it is basically just a no-op.
Copyin(a) support shadow copy ? suppose a is a structure and has a pointer member	This is the "deep copy" discussion that Jeff just went through. Deep copy can be handled but requires a few extra steps.
"When targeting multicore devices (-ta=multicore), you can set the environment variable "ACC_NUM_CORES" " So if i compile it like: pgcc -fast -ta=multicore -o wtv wtv.c && ACC_NUM_CORES=10, it should use 10 threads ?	Yes, assuming the loop you are parallelizing trip count is greater than 10.
Within a single routine, is there any performance difference between using a structured data clause as shown here (enclosing the whole routine) versus an unstructured data enter and exit?	There shouldn't be any difference from the perspective of the performance of the accelerator region itself.

Question	Response
What happens if there is no enough physical device memory?	If you are not using managed memory, you will get an out of memory error at runtime. If you are using managed memory, with proper GPUs and configuration (e.g. linux, Volta, etc.) you may be able to oversubscribe the GPU, i.e. handle data that is larger than physical device memory.
When i used "pgfortran -ta=tesla:cc20 -Minfo=all laplace2d.f90 jacobi.f90", the compiler used managed memory as default because I see messages as "Generating implicit copyout(a(1:n-2,1:m-2))". Is managed option used as default in pgi compilers?	No, you still need to use the managed clause to have managed memory used by default. Looking at a copyout generated by the OpenACC compiler is not a guarantee in this case.
Can I use a global parameters for use in async streams? Such as: "acc ... async(BC_UPDATE)" where BC_UPDATE is defined as: "integer, parameter :: BC_UPDATE=12345"?	Sure, so long as they are integers.
What is OpenACC Slack Link?	www.openacc.org/community#slack
With laplace2d running sequential on my PC i had around 90 secs, i did some loops and data explicitly and run it on GPU, it speed-up to 40-42 secs. I have a tesla=cc20, it is the best speed-up i can get with it ?	Not know the specifics of what you did to the code, nor your system, it's difficult to say. Though, my best guess is that you should be able to improve performance a bit more.
What kind of code can you use on device? Can you use member functions of a struct?	Yes, you can use class member functions in your compute regions. You can decorate the functions with the OpenACC "routine" directive to have the compiler create device callable versions. Note that the PGI will implicitly create device routines for C functions called in compute regions if the function definition is visible during the compilation of source file. This allow the use of templated functions as well where decorating them with directives may be difficult.
I have some comments about Lab 1, is there somewhere I could send them?	yes, please send them to openacc@nvidia.com
Are asyncs on data directives and async on loops share the same streams?	Using CUDA streams is how PGI implements async queues when targeting NVIDIA GPUs. Though other implementations may do it differently.
Sorry I was having trouble getting in ... what was the final decision about the lab?	Are you having issues signing getting into LinuxAcademy for the lab? If so, please shoot an email to support@linuxacademy.com with the subject line of Nvidia Lab and we can help you out.

<#pragma acc parallel loop ...
for (int i=0; i< N; i%2B%2B) {
for (int j=0; j< M; j%2B%2B) {
Anew[i][j] = function of A[i][j-1], A[i-1][j-1], ...
A[i][j] = chage value here too;
}
}
 	
In this case, how to handle the sync. data (A, Anew) between GPU-threads? I mean if A[i][j]
	In this case, the inner loops aren't parallelizable so if you went ahead an forced it to parallelize, then you'd have a data race.

Question	Response
change in another thread may lead an error on another thread due to non-sync.	
Which Spec version we should follow 2.5 or 2.6?	2.6
Question for later (don't want to forget) : What is the defined behavior of an "acc enter data copyin(a)" if "a" is an array that has not been allocated?	This will probably just seg fault at execution when the runtime attempts to copy data from an unallocated buffer.
Last week we saw that -ta=multicore triggers CPU computation wheres -ta=tesla:managed launches GPU computation. However, the loops were parallelized with OpenACC directives. How does the compiler change from OpenACC to OpenMP for CPU ?	The compiler doesn't change to using OpenMP, but I think you mean parallelizing across multicore CPUs which is similar to OpenMP. With OpenACC you are defining which loops to parallelize. This parallelism is general so can be targeted to different architectures.
I am having GTX 1050 TI, then what flag will I be using?	GTX 1050 should be pascal architecture. That would be -ta=tesla:cc60 You can double check me though, if you are using PGI compiler there is a command called "pgaccelinfo" which will give you info (and compiler flags) for all accelerator devices on the system.
-ta=multicore triggers multi CPU computation, but how did the compiler do that if the instruction were OpenACC directives?	The directives are generic and the compiler knows how to translate the parallelism to different targets. The details are a bit much to explain in the forum.
How can I set a number of "threads" for a program with OpenACC directives, like we did with OpenMP on console (export OMP_NUM_THREADS) ?	When targeting multicore devices (-ta=multicore), you can set the environment variable "ACC_NUM_CORES" or, starting in PGI 18.10, call the API "void acc_set_num_cores(int);"
Can targeting GPU automatically handle multi-GPU parallelism? In other words, are the examples illustrated so far only for shared memory parallelism?	Using multiple GPUs requires additional work by the programmer. A single accelerator region will not automatically span multiple GPUs.
Followup: with PGI 18.4 it does not segfault (probably because I never try to use "a"). The use case is that I have conditional arrays that sometimes are used, sometimes not, all within the same structure. For simplicity, I have a single "data copyin()" call for all members, but sometimes some arrays in the struct are not allocated yet. It seems to work now, but I was wondering what the "official" behavior is according to the spec so that the code does not break with future updates/compilers.	You might want to look at using the "no_create" clause for the optional arrays, assuming that you're copying them individually and not the structure as a whole.