OpenACC
More Science, Less Programming

OpenACC Course October 2017
Lecture 3 Q&As

Question	Response
If I have a 3D array in Fortran, how is it mapped in the GPU?	In general, the layout of data structures on the host is matched on the GPU. Specifically, for a 3D Fortran array you would have a single block of memory with 3D indexing. The same structure would be mirrored on the device and there would be no difference in how you would access the array.
Can the pgi profiler be used to profile the cpu and the gpu in the same time to check if cpu and gpu routines run asynchronously?	Yes, the profiler can profile both the CPU and GPU asynchronously. Jeff won't be covering the OpenACC "async" clause today, but will include a link to a web presentation on the subject.
How would I apportion the gang worker vector clauses on a 3D nested stencil loop to map to 3D cuda blocks on a 3d cuda grid? It seems no matter what I try, the compiler never uses 3d blocks and/or grids.	Try using the "tile" clause with 3 arguments where the tile is on a triply nested vector loop.
Is all this supported in gfortran?	GNU does have support for OpenACC including gfortran. Most of the OpenACC standard has been implemented in GNU, though they do lack support for the "kernels" directive. You'll want to use "parallel" if using GNU.
If I have a code that contains loops that have a variable number of iterations with OpenACC pragmas be compiled as a library? and If this library is called from another code, would the compiler be smart enough to optimize the parallelization of those loops?	The launch configuration of the device kernels created from your OpenACC loop is dynamic with the size being used based on the iteration count. So yes, you can use a variable number of iterations. OpenACC code can be put into a library and there'd be no difference in how the loop is schedule between the version in a library and one that is not.
May I access PC equipped with GPU device with CUDA installed on it? Thank you	The Labs do allow you access to a system with a GPU on it, however we don't have systems for you to use outside the lab.
How should I parallelize two nested loops, where I would like to parallelize the top most loop, but keep the inner loop sequential?	That would be dependent upon the loops. If the inner loop is parallelizable and has a larger trip count (like > 128), then you'll more likely what to add additional loop directive on the inner loop so it will be parallelized as well.
Since there is no support for array reductions yet, what is the optimal way to do an array reduction? I have tried serial sums in a parallel loop, and atomic operations (slower). Any other ways?	I'd use a serial sum in a parallel loop over an atomic. Unless the inner loop is small in which case I'd run the inner loop sequentially.

Question	Response
I have encountered problems with OpenACC when I tried to run some examples in a laptop with Nvidia 920 M graphics card. The time was never better than serial version. I think that it is the Nvidia optimus driver. Have you encounter similar problems?	The 920M does not have very many compute resources available so not expected be high performing. The good news is that you can use it for development, but you should use a higher end or newer architecture if you want to see better performance.
Sometimes I get “variable dependence prevents parallelization while using “acc parallel loop”. Why do I get that if I’m promising the compiler that it’s safe parallelization?	It's probably occurring on an inner loop and not the loop where you have the "parallel loop". The PGI compiler will still automatically attempt to parallelize inner loops even if you haven't explicitly added the loop directive.
How does OpenACC work with C++ libraries such as Eigen?	Do you mean can you call Eigen library routines from OpenACC code? The answer is yes, if you have added the "routine" directive to the library which will create device callable code. However, you cannot call host library routines from within device code.
Can you calculate prefix sum (scan) with OpenACC?	No. The reduction operation must be commutative.
Can I use OpenACC in hybrid code, e.g: inside of the node with OpenACC and MPI between nodes? if yes, which compiler can support that?	Yes. Jeff will share some links at the end where you can find additional training on using MPI OpenACC. It works very well and most large applications that use OpenACC also use MPI. This support is independent of the compiler so will work with PGI, GNU, Cray, etc.
Should tile(n1,n2,n3) have the property of n1*n2*n3<max_threads_per_SM?	Yes. So you want to make sure that the product isn't greater than 1024. Better to use literal values than variables so the block size is fixed.
Does collapse() work with a nested loop where there are some calculations outside the inner loop but inside the outer loop?	The loops need to be tightly nested in order to collapse them.
Reduction is needed on each loop?? I have been putting it on my parallel clause and not on the loop clauses within the parallel region - is this incorrect?	Technically, yes. If you want the reduction to go across multiple loop levels, you need to add the reduction clause on each of those loops. PGI does not require this however.
Is it possible to use two or more GPUs within the same server? Does the compiler disseminate jobs to more than one GPU?	Yes. You can toggle between GPUs from within an OpenACC code, though it can be cumbersome and not automatic. It recommended to use MPI OpenACC when using multiple devices.

Question	Response
Is the -ta=tesla option used for all others nvidia architectures (kepler, pascal, ...)?	The default targets will depend on the compiler version you are using. The current PGI 2017 compilers use CUDA 7.5 by default so "-ta=tesla" will created device code for Kepler and Maxwell. For Pascal, you need to use "-ta=tesla:cc60" and/or "-ta=telsa:cuda8.0". Once we move to using CUDA 8.0 by default, then Pascal will be included in the default list. For Volta, you'd use "-ta=tesla:cc70" or "-ta=tesla:cuda9.0"
AMD GPU has equivalent of CUDA managed memory; can OpenACC use it?	It would be up to the OpenACC implementation on whether to take advantage of this support. The OpenACC standard does make data management optional so there's no reason why an implementation couldn't use it, but that's not part of OpenACC itself.
Where to find information on multi-GPU support for OpenACC?	Here is the link to the talk covering multi-GPU support: http://on-demand-gtc.gputechconf.com/gtc-quicklink/hhZdn
Is there a free compiler that support OpenACC?	Yes, there is a GCC compiler and PGI Community Edition compilers that are available for free
Are there any publications/studies that compare the performance of OpenACC code compiled with PGI vs. GCC compiler? I suppose PGI will perform better, but it also comes at a price in a production code using GCC as you have to port the code to a different compiler that could have hick ups in certain 3rd party libs (e.g. boost library). In other words, such a comparison would help weighing performance vs. developer effort.	I don't know of any studies that did this. The GNU folks have shown me some benchmark results where they were able to match PGI, which is good. The major caveat with GNU is that they don't support the "kernels" directive, so you are limited to using "parallel". So provided that you can put in the extra effort to guide the GNU compiler on how to parallelize your loop, you should be able to get decent performance from GNU.
I am trying to optimize a 3D stencil triple loop for GPU. I have been playing with gang, worker, vector, tile, etc and so far the fastest is just using "loop,loop,loop". Is there a standard stencil example that has been tested on Psacal GPUs to give some hint on what the optimal clauses should be?	I do have some stencil examples but not sure if it would apply to your program. Do you mind post this question on the PGI User Forum including a code example of what you're trying to do? It will easier to answer there.
Is the cache directive supported in PGI 17.9? I put in in a parallel region but do not see anything in the compiler output recognizing it...	Yes, it's supported though still tricky to use. Also, the compiler doesn't emit a feedback message for the cache directive, so it may be using it but just not telling you.
How can one in OpenACC parallelize across GPU devices?	The easiest way is to use MPI OpenACC with one MPI rank per GPU.

Question	Response
would I see the cache working in the pgprof output? it would be in the shared memory per block?	Look at the output form "-ta=tesla:ptxinfo" for the amount of shared memory used or review the generated device code (.gpu file) via "-ta=tesla:nollvm,keep".