OpenACC

More Science, Less Programming

OpenACC Course October 2017 Lecture 2 Q&As

Question	Response
Is there any plan to support Intel integrated GPUs?	OpenACC itself works fine for Intel iGPUS, but I'm not aware of any compilers that currently support it. It's possible that GCC does or will support this, but I'm not certain.
For what kind of problems is CUDA better than OpenACC?	CUDA exposes the low-level details of the GPU architecture, so you can often tune a CUDA kernel more than you can an OpenACC loop. Codes that rely a lot on the GPU's shared memory or registers to communicate among threads often work better with CUDA, because OpenACC doesn't expose those low-level details. Frequently when you have loops with a lot of code in them, a CUDA programmer may be able to better manage how that code is used on the GPU and will be able to outperform OpenACC. My usual guidance is to start with OpenACC, get on the GPU quickly, and then when you find a particular part of the code isn't performance as well as you'd like, you can write that part in CUDA and leave the rest in OpenACC.
A couple of question related with the first lab: 1) why the code runs slower when using 8 cores compared with 4 cores execution. 2) Why the code run faster in C than in Fortran?	The CPU on the lab system only had 4 physical cores with 2 hyper threads per core. Hence when moving to 8 OpenACC cores, you over saturated the CPU cores so didn't see any additional speed-up and even can slow down the code.
In OpenMP there are different parameters such as PRIVATE, SHARED, etc, do we need to do anything similar with OpenACC?	In OpenACC you may need to use private sometimes, but everything is assumed shared by default. Privatization is sometimes required in order to get correct results. If you're using OpenACC's PARALLEL directive, as opposed to the KERNELS that John has been teaching, then you may need to look at the REDUCTION clause as well.
Is there any disadvantage using OpenACC with CPU (vs OpenMP)? What do data clause do for CPU?	What we've seen in that most of the time the OpenACC compiler will do just as well as OpenMP on the CPU, but there's exceptions. OpenACC has no way to control which threads run on which CPU cores (thread affinity) like OpenMP does. So in cases where you're on a machine where thread or memory affinity really matters, OpenMP has a way to handle that and OpenACC does not. OpenMP also gives you a bit more control about how to map your loop iterations to the threads by using specific schedules, so for very load-imbalanced loops OpenMP will allow you to do better. For the average loop where all of the iterations are doing the same thing and are really parallel, the two should perform comparably.
Using OpenACC with multicore mode is identical as using OpenMP?	Very close, yes. With OpenMP you're taking a lot of control and telling the compiler how to parallelize the loop to threads. With OpenACC you give the compiler the opportunity to try to use its smarts to do better. But, the directives are very similar. The advantage with OpenACC is you can write the directives, get good performance on the multicore CPU and then take the exact same directives and it'll run well on a GPU as well. With OpenMP you need different directives if you're running on a CPU or GPU (or something else altogether).
Why isn't there an almost linear speedup for the 2000X2000 case using 4 cores?	I haven't explored exactly why the 2000x2000 case doesn't scale as well as the 1000x1000 case. However, the 1000 case does fit in the L2 cache while the 2000 case does not. I believe the lack of scaling in the 2000 case has a similar cause.
What kind of gpu card you have on your system?	The Lab system has an older Kepler (compute capability 3.0) card.

Question	Response
Was total 40 seconds here?	Yes, the initial port of the OpenACC when moving to the GPU is quite slow. As John explained, the vast majority of time here is being spent managing the data between the discrete CPU and GPU memories. This is why we need to add data regions.
it seems you still haven't answered question 2. Oftentimes that is because a C compiler ca apply optimizations. Could be that the Fortran compiler doesn't apply optimizations	That's generally not true. Compilers can usually apply more optimizations to a Fortran code than a C code because of the rules the language imposes on the programmer. Sometimes a C compiler can do better, but usually not. In this case I think there's an error in timing or building that resulted in worse performance on the Fortran version. Based on experience with this code and others like it, the performance should be virtually identical.
yes, compiler flags. That's what I mean with "optimizations"	Right, I think a compiler option was probably missed. The preferred option with PGI is '-fast', which goes above and beyond something like -O2 or -O3 and picks what PGI thinks is the best flags for your machine.
How do you identify if your problem does or does not fit within the L2 cache?	By the array size. In the 1000x1000 case the arrays are about 16MB, and 64MB in the 2000x2000 case. The L2 cache size on the Lab system is 20MB (as seen by "cat /proc/cpuinfo").
On the slide for Exercise 1: C Solution, why does the second kernel require an extra set of braces, while the first doesn't?	Both are correct. The first kernel will use the structured block of the for loop while the second defines it's own structured block. If you want the kernels region to span multiple loops, then you will need to use the secondd method of defining a structured block.
Any plans to support OpenACC in Julia?	Good question. We had a discussion recently on the OpenACC technical committee about what other languages we may be able to support. Right now I'm unsure of whether Julia has a notion of pragmas or directives like C/C and Fortran do, but it's on my list of things to look into.
From which compute capability can you use your nvidia GPU?	I believe the PGI compiler supports as far back as cc 2.0 and supports up to the latest GPUs, which is cc7.0. At some point that will probably change to 3.0 and beyond.
Did I understand correctly that OpenACC's explicit strategy is to focus on offload (as opposed to shared memory)?	OpenACC's strategy is to give the programmer the ability to parallelize their code for any parallel machine, shared memory, multicore, GPU, whatever, using the same set of directives. OpenACC doesn't care if it's an offloading machine or not, although your code will be more portable if you assume it may be run on a offloading machine because it's easier for the compiler to ignore your data directives than to implicitly figure them out for you.
Is it possible to get the same info that it's showing in pgprof from gprof (or any other GNU-based tool)?	For the CPU side, gprof can get you similar information as pgprof. Though unless something has changed recently, I don't believe gprof has been updated to include GPU profiling support.
I think there's an error on the slide titled "ARRAY SHAPING". Shouldn't the C pragma have copyout(b[s/4:s/2]) instead of s/4:3*s/4? Either that or the Fortran one ought to be copyout(b(s/4:s)), right? Thanks.	Not necessarily. In C, (start:count) is used. So "copyout(s/4:3*s/4)" is saying start at element "s/4" and copy "3*s/4" elements. If s=12, then this would be start at element 3 and copy 9 elements . In the Fortran case, (start:end) is used so this would be start at element 3 and end at element 9.
Are AMD GPUs supported by OpenACC?	At one point in time both the PGI and PathScale compilers supported AMD GPUs, but I don't believe they do now. I've spoken with someone at AMD in the past and I think they're hoping to begin supporting OpenACC again soon.

Question	Response
Can you give us a fast comparison between what you can get from pgprof and scorep/scalasca/vampir? (I'm a score-p suite user)	PGPROF is based on NVIDIA's Visual Profiler, so it will give you more GPU-specific details then the alternatives you listed. These alternatives do a much better job of showing the big picture though, particularly MPI communication and how it interacts with the GPU.
Last week I did all the thinking before the lab...I'm happy with that because there were hints in the profiler that wold have told me without thinking. Should I do a lot of thinking before opening the lab again or is the lab set up so I will iteratively trial-and-error solutions? Hope the question makes sense	The intent is that Lab2 is set-up to lead you through the process of profiling your code to understand where the performance problem is (data movement) and what directives you need to add where. We did include a final solution this time if you get stuck.
Trailing my previous question: is is possible to download the info from the execution and analyze it locally on the server using pgprof?	Yes. In Lab2, this is the method we use. You first save the profile by using the command line profiler (pgprof -o myprofile.prof) and then import the profile into the GUI profiler. I often will profile on a remote system, copy the profile to my local workstation for visualization.

what if I am working in a large code where most of the time the data needs to be in the gpu... can I create a data region from which I call subroutines that are written with openacc kernels?
 	
$acc data (x,y) {
call subroutine1(x,y)
call subroutine2(x,y)
call subrountine3(x,y)
}
	Yes, absolutely and this is very common. Many codes have the KERNELS regions inside of function calls and the data region way up in the main program.

Will the PGI Community Edition always remain free?	Yes, it is free for everyone.
In FORTRAN 90, I want to communicate a subset of an array between separate subroutines, in a complicated code. The subset of the array data will not change during the kernel region. I don't want to explicitly communicate it through subroutine input, as parts of it are used in different places. Can modules be used for that? What is the best approach for OpenACC in this situation? I am very familiar with OpenMP or OpenACC.	In this case, it sounds like you may wish to use module data and use the ACC DECLARE directive to make that data remain resident on the GPU for the extent of your run.
Does the openacc kernels that run in the GPU allow to run something simultaneously in the CPU?	You can run on both the CPU and GPU simultaneously, but KERNELS won't do that automatically for you. You can use the ASYC clause to send work to the GPU, do other work on the CPU, and the use the WAIT directive to get them back in sync.

Question	Response
What would happen in runtime if the data needed to move into GPU is larger than the memory it could offer? Any error handling mechanism?	Ultimately, data all device management translates to the CUDA API. If you need more memory than the GPU has, cudaMalloc will thow an error.
How does OpenACC deals with memory transfers between devices within the same node? Does it follows a scheme Device1->Host->Device2 ? or directly like Device1 -> Device2?	OpenACC itself does not manage memory between multiple devices. Though if you are using a CUDA Aware MPI then you can use this transfer directly between devices.
Could you comment on what would happen in the lab if we removed from laplace_kernels.final.c the kernels directive in the initialize function? (I would like to try it, but can't seem to obtain a python kernel for my notebook)	That would be fine depending on where you put the data directive. If it's before the initialize routine, be sure to use the "update" directive to set the values in the device copy of the array. Or move the data region after the initialize routine and use a "copyin" clause.
If I'm using P100 GPU, do I use the same "tesla" option or a different one?	With PGI 2017, we default to using CUDA 7.5 so don't target P100 with "-ta=tesla" since P100 requires CUDA 8.0. For P100, use "-ta=tesla:cuda8.0" or "-ta=tesla:cc60".
where can I find your Tshirts? :-) Thanks for the tutorial	come to SC17 User Group meeting :)
In the pgprof GUI section, where is the profile file sitting?	It's under "File System" the the "notebooks" directory.