OpenACC Course
Office Hour #2 Q&A

Q1: How many threads does each GPU core have?
A: GPU cores execute arithmetic instructions. Each core can execute one single precision floating point instruction per cycle for one thread. For example, a GPU with 192 CUDA cores can execute 192 single precision floating point instructions per one cycle.

Q2: Warp size is 32, i.e 32 threads per warp. Can we launch 32 threads concurrently?
A: Yes, warps are executed in lock-step, meaning each instruction is issued and executed for all 32 threads in a warp concurrently. GPU can easily switch between warps to hide latencies for arithmetic or memory instructions. In order to effectively use a GPU, it's necessary to expose more than just 32 threads worth of work, but more like 10,000 threads worth of work.

Q3: What about more tightly coupled calculations? For example, will a molecular force field, run faster on a CPU or GPU?
A: It depends on the problem, but in general molecular dynamics calculations run very efficiently on GPUs and can outperform CPUs. You can check performance results for popular molecular dynamics packages like NAMD, AMBER, LAMMPS, and others.

Q4: Can one run Visual Profiler on nonlocal machines?
A: It is possible to run Visual Profiler remotely using remote access clients like NX. Alternatively, you can collect information using nvprof on a remote machine and view it locally with Visual Profiler. This can be done by first collecting the timeline remotely (`nvprof -o timeline.out ./a.out`) and then using File -> Import to bring it into Visual Profiler on your local machine.

Q5: Can you use multiple GPUs with OpenACC WITHOUT using the API and just with directives? (At least for a "double" card such as a K80?)
A: No, currently you have to use runtime library API calls to select GPUs, such as acc_set_device_num. You can also select GPU number at runtime by setting environment variable ACC_DEVICE_NUM and use different CPU processes per GPU.

Q6: Slide 26: 2 loops are involved in the same pragma acc kernels directive. In the last homework, the solution used one pragma for each loop. What is the advantage of separating the loops?
A: Both approaches will create two kernels, one for each loop. With kernels directive compiler will analyze the code and try to identify the parallel regions (it may fail to do so), while parallel loop directive is developer's guarantee for the compiler that the following loop is safe to parallelize. Check slide 47 of lecture 2 for more differences between kernels and parallel loop.

Q7: In general, why arrays cannot be defined in device code?
A: Arrays have to be declared outside of parallel regions, otherwise they will be treated as private for each thread. You can use small private arrays in parallel loops but they have to be allocated on stack. Dynamic allocations within compute regions are not supported. Declaring a lot of private arrays (or even a single very large array) will use up the limited memory available on the GPU compute resources.

Q8: Could you please weave "private" and "reduction" into a transparency in the next lecture?
A: When a variable is declared as private a unique copy of the listed variables will be created for each iteration of the affected loop to prevent two iterations running in parallel from both reading from or writing to each other's copy. Loop induction variables ("i", "j", "k", etc.) are made private by default. The reduction takes this a step further and uses the specified operation (+, *, min, max, for instance) to reduce from all of those unique copies to one value that will stored in this variable at the end of region.

Q9: Does reduction+ support C++ overloaded operators?
A: No. Reduction clause only supports numerical data types in C/C++ (int, float, double, complex).

Q10: Does OpenACC support parallel primitives like scan?
A: For parallel primitives we recommend to use Thrust, CUB or CUDPP libraries. They are completely interoperable with OpenACC.

Q11: For the case of a loop on a vector of structs, which requirements must be fulfilled by an struct, to make possible to accelerate the loop using OpenACC?
A: In general there are no particular restrictions. However, if the struct contains dynamic arrays or pointers to other data structures then it would be necessary to use Unified Memory or perform manual deep-copy of the structure to use it on device. Unified Memory can be enabled using compile option -ta=tesla:managed. For manual deep-copy it would be necessary to use runtime API calls. Future compilers will improve support of deep copies of data structures.

Q12: Does OpenACC support dynamic structures?
A: With OpenACC and Unified Memory, any memory allocated with malloc or new can be accessed by GPU and used within OpenACC parallel regions. Alternatively, you can use data directives to control data movement which will be covered in the next lecture. You cannot resize data structures within parallel regions.

Q13: Is the code currently discussed already available?
A: Yes, the code for lectures and labs is available on github: https://github.com/NVIDIA-OpenACC-Course/nvidia-openacc-course-sources

Q14: Do you need a GPU to run the code or is there an emulator?
A: There are no GPU emulators officially supported by NVIDIA, so you need a GPU to run the code. You can also use the new PGI x86 OpenACC option, you don't need the GPU, you can use the ta=multicore.

Q15: The bus used for migrate data from GPU memory and System Memory, will be dedicated of just we can use a normal bus?
A: PCIe is used for data migration between GPU and System memory, it's a normal bus which is used for other communications between CPU and GPU as well.

Q16: But C allows for function arguments to point to the same memory, so there could be dependencies if the routine is called with arguments that overlap. Will the speaker address this? (Fortran has a different story.)
A: Yes, this issue has been addressed on slides 35-36. It is necessary to use restrict keyword in C to specify that pointers/arrays will not alias.

Q17: slide 32: why C, A and B cannot be defined in loopBody function?
A: If arrays are defined in loopBody function then they will be treated as private for each thread.

Q18: Does nvprof uses sampling or just-in-time instrumentation, or a combination of those?
A: nvprof uses sampling and performance counters available in GPU.

Q19: Is there any compatibility issue using "hybrid" GPU (quadro)?
A: PGI compilers support most CUDA-enabled GPUs with Compute Capability 2.0 or newer, including Quadro GPUs

Q20: Is it possible to work on the lab2 without a managed memory device capabilities?
A: Yes, it is possible. The only difficulty here is you will have to control all the data movements yourself.

Q21: You just mentioned unified memory in conjunction with OpenAcc. This confuses me - I thought unified memory was for CUDA only
A: Unified memory is not CUDA specific because it is also exposed in OpenACC if you specify managed compiler options. Unified memory is implemented through the CUDA driver and hence can be used in both OpenACC and CUDA. There is a blog post talking about how to use unified memory with OpenACC: http://devblogs.nvidia.com/parallelforall/combine-openacc-unified-memory-productivity-performance/

Q22: How do I use both OpenACC pragmas and CUDA kernels to process the same data?
A: There is an API available where you can get a GPU or host pointer for a particular array and can use them to write your kernels.

Q23: My principal practical interest in OpenACC right now is accelerate a part of my code in which I have four loops running over the elements of a STL::vector of structures, each structure is accomplished by various integers (5) and two arrays of doubles. Is possible to use OpenACC to speed up my computations?
A: Since you have loop and each loop goes over an array of structures - then yes. Because you have some nested arrays within your structures, the easiest way to run your code is through unified memory and OpenACC. The next step check if you need to manage memory explicitly through memory copy OpenACC API.

Q24: You said that OpenACC for default use only one GPU. Is it also a case if we connect GPUs with SLI?
A: It doesn't matter, because the way the OpenACC sees devices even if they are connected through SLI which is not used in compute at all, OpenACC will see them as 2 devices. So you will have to switch between 2 devices if you want to use both of them.

Q25: Hi, in the lab2 there are 2 OpenMP pragma nested loops, where run these loops? CPU or GPU?
A: If you put an OpenACC directive, these loops will run on GPU. If you use OpenMP – they will run on CPU unless you have an OpenMP GPU enabled compiler.

Q26: Does unified memory cause a very large usage of virtual memory with openacc?
A: Unified memory on Kepler and Maxwell architectures – it will pin the memory and will be using some memory that you won't be able to use otherwise. It will not use more memory than you allocate in your code.

Q27: Is possible to use data types from Armadillo library with OpenACC?
A: It should be possible to use it, but we haven't use this library here – so can't guarantee.

Q28: What about device memory debugging? Is there any free tools that are available?
A: Yes, there are tools that allow you to do debugging in a GPU application. If you run on Linux, the best tool will be CUDA-GDB, it is free and it comes with the CUDA toolkit. If you use Windows – Nsight Visual Studio edition.

Q29: Is there a way to tell OpenACC to use shared memory versus more cache?
A: In Kepler architecture there is a way to specify 48 or 16kb of shared memory vs. L1 cache. Not sure if it is possible to do in OpenACC, but you can always use CUDA API for that.

Q30: Can we use one GPU along with multi-core CPU?
A: Yes, you can use CPUs and GPUs together. In OpenACC there is an async close that allows not to wait for the talk completion and run tasks at the same time.

Q31: Is there any profit in using 1D indexing of array elements (A[i+size_i*j]instead of A[i][j]) in comparison with the lab examples?
A: You could use 1D indexing of array elements. Sometimes though it can make things worse for the compiler because it can't understand if there is any dependencies. It might help to specify a loop independent close. But there is no benefit really in using 1D indexing.

Q32: Can you comment on OpenACC performance vs GPU hardware, such as in the table at the end of the lab. In there a minimum GPU model to achieve good performance.
A: In general OpenACC can achieve pretty good performance, even close to CUDA performance if the code is relatively simple (no complicated mapping for example).

Q33: Will the kernel performance go down if kernel is to long? Is there an optimal size of the kernel?
A: If the kernel is too long it won't necessarily slow down your application. However there is always a tradeoff – you can split the kernel into multiple parts, but then you have to watch kernel launch latency. In general if you expose more threads within kernel it is always good. Also to utilize a GPU efficiently you need to have at least 100 threads, but it always depends on the actual GPU.

Q34: In this great Nikolai's blog post, what's the meaning of "On systems with multiple GPUs, Unified Memory allocations are visible to all devices with peer-to-peer capabilities"? With multiple GPUs, refers to SLI GPUs?
A: When using unified memory with multiple GPUs, you need to have GPUs connected through P2P which usually means thought the same PLX PCIe switch. Without P2P the performance might slow down.

Q35: What about others languages is available for c++ and fortran only? Will you have plans to java support?
A: We support Python you can use with GPUs, there is also a wrapper for Java.

Q36: Is CUDA-GDB capable of tracking structure pointers passed on the GPU using OpenACC API?
A: You should be able to get some break points where all the registers etc. Please also check a PGI DBG debugger for OpenACC.

Q37: Is there a way to override any use of OpenACC in code. Imagine a case where you use a large code and you want to combine it with your own code that also uses OpenACC, but don't want its use of OpenACC be allowed.
A: Any code that you compile without –acc will not be using OpenACC even if you have OpenACC pragmas in there.

Q38: With OpenMP you have regions, so you can use that library to have threads and then omp_locks. Does OpenACC only offer parallelization, or does it also provides independent threads?
A: In OpenMP you have all the different locks and barriers you can setup, and in OpenACC they are hidden. This is the main difference between OpenACC and OpenMP.

Q39: Can we submit a job using two executables, one is built for GPU, the other built for CPUs using MPI directives?
A: Yes, you can. The way you do this – you can have an if statement controlling which GPU or CPU work to run. You don't need to have 2 executable for that.

Q40: Can we use the executable built using OpenACC directives and OpenACC flag on CPUs without GPU?
A: Yes, you can just use it on x86 multicore. OpenACC works on both CPUs and GPUs.