Scalable Node Programming with OpenACC

Date: September 20, 2017
Presented by: Michael Wolfe, NVIDIA

The presentation material along with the video will be made available at the following link:
https://exascaleproject.org/event/openacc/

Q. For C++ using PGI compilers, what is the best dynamic array type to use? Can one use C++11 vector<>, boost/Eigen? new Array? or malloc?

A. Avoid std::vector since it's not thread safe and given it's an opaque type, the compiler has no visibility as to what the actual data type is so doesn't know how to copy it on the device. You can use CUDA Managed memory (-ta=telsa:managed) to help since you don't need to worry about the dynamic data movement. Still, std::vector isn't the best choice for highly data parallel architectures such as the GPU. New or malloc'd arrays of fundamental data types work best.

Q. In an inner nested OpenAcc parallel loop one can use counters (say an int), but how does one add to a common array for results?

A.

```
For i
  For j
    For k
      Arr[i][j] …  // this isn't parallel due to a dependency over the k loop
```

We could solve this using atomics or possibly using a scalar reduction over the k loop and then assign the result back to arr.

Example using a reduction:

```
For i
  For j
    Sumval = Arr[i][j]
#pragma acc loop reduction(+:Sumval)
    For k
      Sumval += ….
    End for k
    Arr[i][j] = Sumval
```

Example using Atomics:

```
For i
  For j
    For k
#pragma acc atomic update
      Arr[i][j] = Arr[i][j] + ….
```

Q. For the CPU openacc code, does the compiler optimize it for one core using SIMD like "omp simd"?

A. The compiler may auto-vectorize the code using SIMD instructions. Though the PGI compiler doesn't currently doesn't utilize the "vector" schedule clause (which is roughly equivalent to "omp SIMD") and instead relies on auto-vectorization. We hope to extend this capability in the future.

Q. Is cloverleaf an explicit algorithm or does it use CG/multigrid?

A. Its it's own application and can be found at: http://uk-mac.github.io/CloverLeaf/

Q. Prior to 17.7, is there no way to do manual deepcopy? Are structures just not supported or just hard to implement?

A. PGI has supported manual deepcopy since 2014 and it is being added to the OpenACC 2.6 standard. In PGI 17.7 we introduced implicit deepcopy (i.e. the compiler performs the deepcopy) for Fortran UDTs. Since Fortran arrays have descriptors, the compiler runtime has enough information to perform the implicit deep copy. For C/C++ aggregate types with dynamic data members, the compiler runtime has no information about the size and shape of the dynamic members (they're just unbounded pointers), so can't implicitly make a copy.

Q. For manual deepcopy, can the data enter clause for the structure and its children be in the same clause, or does one need to use separate clauses for each level?

A. They can be in the same clause, but currently for PGI, order does matter so the parent should come before the children. I.e. "enter data create(v,v%r[0:n],v%t[0:n],v%p[0:n])". If "v" (the parent) isn't on the device yet when "v%r" is created, the compiler won't be able to "attach" it to the parent. So "v" must be created before "v%r"

Q. Can I use two GPUs in a single node using a single unified memory?

A. The short answer is yes. Though only with more recent versions of CUDA (8/9) and recent devices Pascal/Volta. More details can be found at: https://devblogs.nvidia.com/parallelforall/unified-memory-cuda-beginners/.

Q. Does deepcopy in 17.7 support the "default(present)" clause?

A. These are related but different things. “default(present)” states that all variables that are implicitly shared be tested that they are present. Deepcopy has to do with how the data is created and copied to/from the device. A variable that has been successfully created on the device (deepcopy or otherwise) will have an entry in the present table and therefore can be tested for presence on the device.