1. Introduction

OpenACC is an Application Program Interface that contains compiler directives for C, C++ and Fortran code to offload from a CPU to an attached accelerator. OpenACC allows the programmer to write GPU code without handling low-level tasks such as data-copying and thread management. The interface of OpenACC is very similar to OpenMP and should be familiar to those who have worked with the latter.

2. Syntax

OpenACC uses higher-level pragmas to tell the compiler identify and handle parallelization on behalf of the programmer, the primary pragmas and clauses are listed below.

a. **kernels** pragma

   This pragma can be used to tell the compiler to attempt to identify potentially parallelizable code in the following block of code. If the compiler decides that there are dependencies that prevent safe-parallelizing of the code it will ignore the pragma and execute the block serially.

b. **parallel** pragma

   The parallel pragma works differently from the kernels pragma in that the former tells the compiler that the following block of code is parallelizable, while kernels pragma tells the compiler there *might* be parallelizable code. It is safer for the programmer to use the kernels pragma if they are unsure if a region is parallel-safe, though kernels can still miss potential dependencies.

c. **data** pragma

   The data pragma tells the compiler we want to move data between the host and device/s using various clauses for data movement. Following are common data clauses, all of which use a list of data as a single parameter.

d. **copy** clause

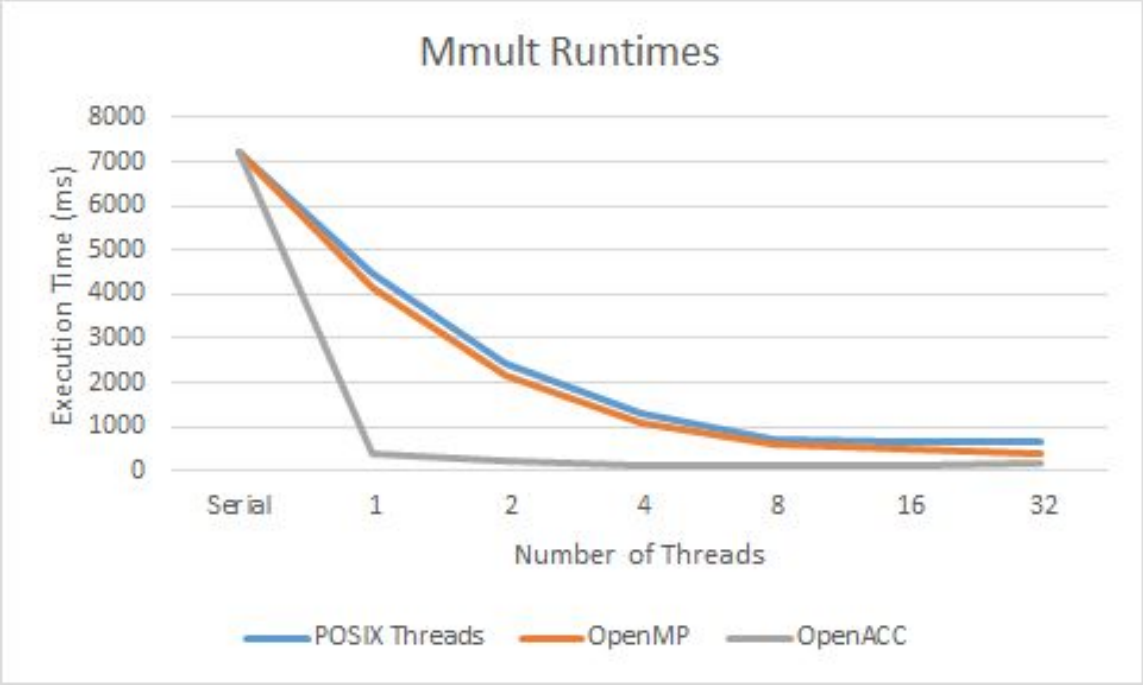
   Allocates memory on the GPU device, then copies data from host CPU to GPU device when entering a parallel region. Upon exiting the parallel region, copies data back to host.

e. **copyin** clause

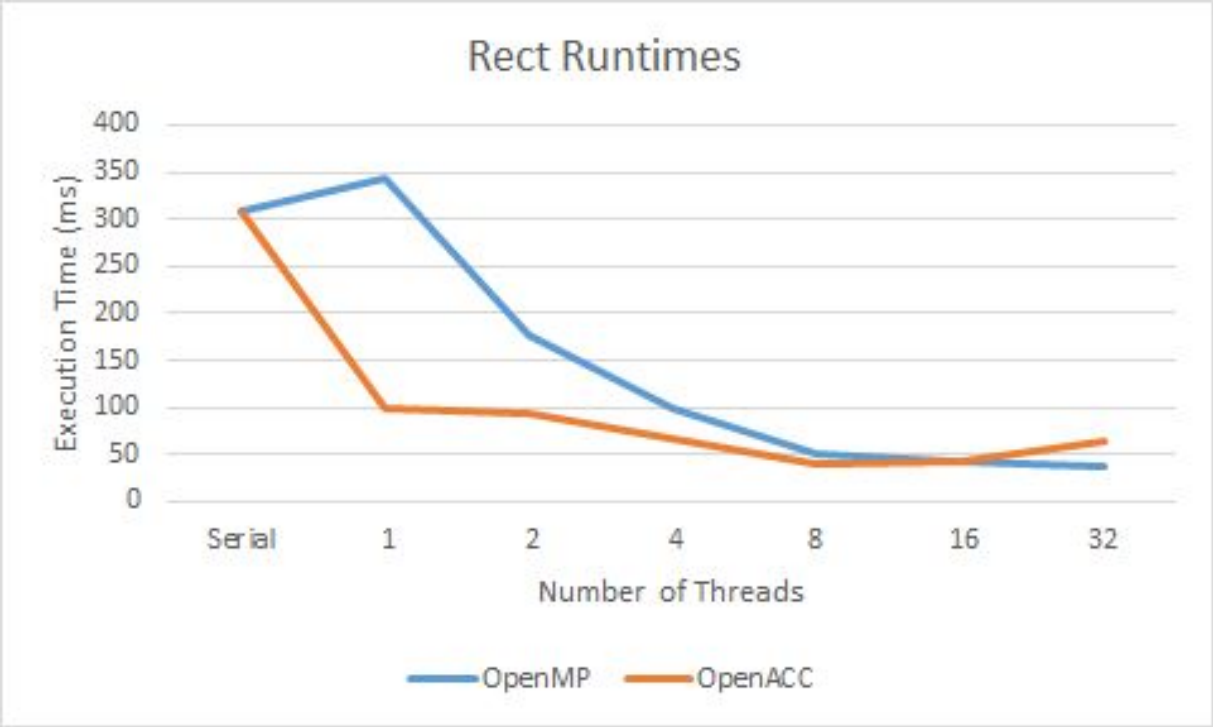
   Allocates memory on GPU device, then copies data from CPU to device upon entering the parallel region.

f. **copyout** clause
   Allocates memory on GPU device, then copies data to the host upon exiting the parallel region.
g. **create** clause
   Allocates memory on GPU device, does not copy data.
h. **present** clause
   Notifies that data is already present on GPU device from another data-clause region.
i. **loop** clause
   The loop clause is similar to the 'for'-clause in OpenMP, with the exception that the loop is executed on the GPU.

3. Comparison



OpenACC performs increasingly well on the matrix multiplication problem, with a single thread offering massively increased performance over both Pthreads and OpenMP. The performance of OpenACC remains faster than both Pthreads and OpenMP with any number of threads; noting this, the OpenACC library is most likely implementing significantly more optimization on parallelizable code than either of the former libraries.



For an additional experiment, we compared OpenMP and OpenACC on the Riemann sum for rectangle approximation (rect). The chart above shows that OpenACC again offered a large advantage on lower number of threads over OpenMP, however the fastest time on the trials occured with OpenMP running on 32 threads. While the overall difference between OpenACC's time at 8 threads and OpenMP's fastest time was ~3 milliseconds, OpenACC did achieve this with considerably fewer threads than OpenMP.

4. Discussion

a. What was the hardest part of the project?

The decidedly hardest part of getting OpenACC to work is using a compiler that is formatted to use it, as the GCC distribution used on the linux cluster is not compatible with OpenACC. In order to compile and run the implementations, our group needed to use the engineering crc cluster, which only a single member of our group had a login for. While we could install GCC on our personal computers, it is a large and complex installation, for which it states during installation that it should only be carried out by those who know explicitly what they are doing.

b. Is this library better than Pthreads and OpenMP?

This library does offer a large performance advantage over Pthreads and OpenMP in runtime with almost all values of threads, along with having a decrease in performance occur at larger numbers of threads then either Pthreads or OpenMP. While this is the case, the complexity of the system is far more difficult to use than OpenMP, although it has more versatility than the latter.

c. How much better/worse in terms of performance?

Running with lower-numbers of threads, the performance increase in OpenACC over OpenMP and Pthreads is remarkable; for a single thread in the matrix multiplication problem OpenACC completed in under a tenth the time it took either OMP or Pthreads. The Riemann Sum algorithm also ran considerably faster on OpenACC than OMP. For higher values of threads in the matrix multiplication problem OpenACC also beat out both other libraries by a considerable factor, with Pthreads and OpenMP approaching OpenACC's time on one thread themselves using 32.

d. How easy/difficult was the implementation of a parallel program?

OpenACC's syntax is very similar to OpenMP's, and basic parallelization of programs is as relatively easy as it is in OpenMP. However utilizing the data movement clauses and pragmas to create a more efficient parallelization is more difficult than utilizing some of the clauses in OpenMP.

e. Is it a good idea to add the language as part of the class?

This language could possibly be included as part of this class if the linux cluster's GCC version is updated to version 9.2, which is compatible with OpenACC. However, the concepts within the language are a relative mix of OpenMP and MPI, the former with the utilization of pragmas and the latter with data movement. This being the case, OpenACC is still far faster than either OpenMP or Pthreads in the tests we conducted, and could prove to be beneficial in demonstrating a modern efficiency boost from a parallel library.

5. Conclusion

Overall OpenACC is a powerful parallelization library that offers staggering performance increases on the algorithms we have been experimenting with in class (namely mmult and rect). The ability to parallelize code as easily as with OpenMP while having the ability to explicitly move data as with MPI is a powerful tool to fine-tune algorithm efficiency gains. The library is tailored to a wide range of programmer experience; it offers both options such as the *kernels* pragma which is forgiving to those less familiar with parallelization, and such as the various clauses in the *data* pragma which allows a more hands-on approach. This library is worth the difficulty in setting up as it offers a wide range of options and large performance increases over both the POSIX-threads and Open-Multiprogramming libraries, and would be a beneficial addition to the high-performance computing curriculum.