OpenACC

Birgitte Brydsø

HPC2N, Umeå University

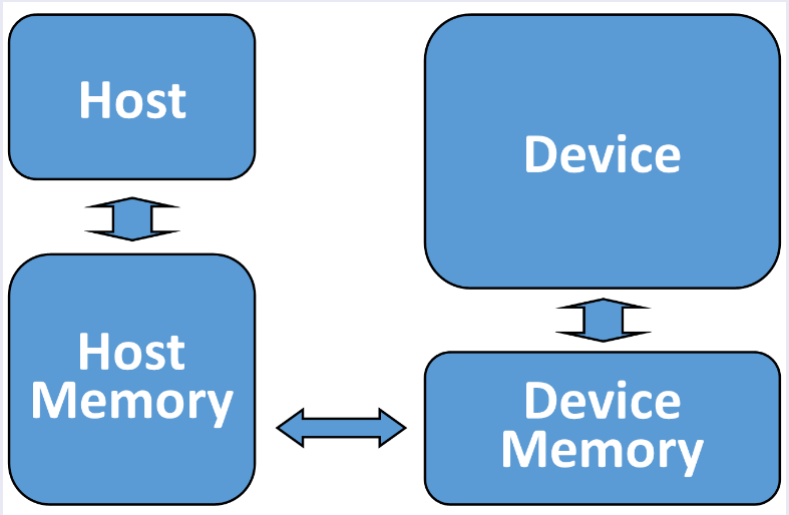
12 December 2017

OpenACC

Overview

What is OpenACC?

1. a software accelerator that offers portability between compilers
2. a programming standard for parallel computing developed by Cray, CAPS, Nvidia and PGI
3. designed to simplify parallel programming of heterogeneous CPU/GPU systems
4. Like OpenMP, it is compiler directive-based - C, C++ and Fortran code can be annotated to identify areas to accelerate using compiler directives and additional functions
5. works on Nvidia, AMD and Intel accelerators
6. works for PGI and Cray compilers - and mostly for GCC



The OpenACC Accelerator Model

- OpenACC supports offloading of both computation and data from a host device to an accelerator device.
- These devices may be the same or may be completely different architectures (like a CPU host and GPU accelerator)
- The two devices may also have separate memory spaces or a single memory space

Steps to add OpenACC to your code

1. Include the OpenACC header file
   - **C**: `#include "openacc.h"`
   - **Fortran**: `use openacc` or `#include "openacc_lib.h"`
2. Analyze code to determine which areas would benefit
3. Add compute directives
4. Add directives to optimize data movement
5. Optimize your application using kernel scheduling

- Identify high-level, expensive loops
- Place OpenMP directives on high-level loops
- Vectorize low-level loops
  - Eliminate dependencies
- Add OpenACC directives now when OpenMP parallelism and low-level vector parallelism is exposed

Grammar

- All openACC directives start with
  - **C**: `#pragma acc`
  - **Fortran**: `!$acc`
- This is followed by the directive name and an optional list of clauses.
- Most directives are followed by a structured block.

Extensive list of pragmas (directives) (Fortran in parentheses)

- Define parallel computation kernels to be executed on the accelerator
  - `#pragma acc parallel` (`!$acc parallel`)
  - `#pragma acc kernels` (`!$acc kernels`)
- Define and copy data to and from the accelerator
  - `#pragma acc data` (`!$acc data`)
- Define the type of parallelism in a parallel or kernels region
  - `#pragma acc loop` (`!$acc loop`)
- Other
  - **Fortran**: `!$acc directive [clause [,] clause] ...`
  - Often with matching end directive around structured code block `!$acc end directive`
  - **C**: `#pragma acc directive [clause [,] clause] ...`
  - Often followed by a structured code block

OpenACC

#pragma acc parallel (!$acc parallel)

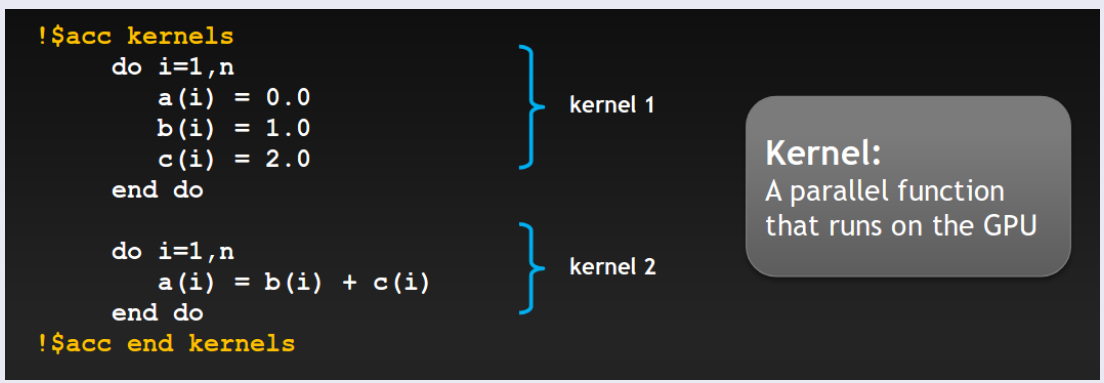
`#pragma acc parallel (!$acc parallel )`

- Tells the compiler to parallelize the code block.
- Compiler can decompose however it feels is best
- Gangs of workers created to run the code in the block.
- Code not in a loop is run in gang-redundant mode (execute same code across all gangs)
- Parallelism achieved in loops by splitting work among several gangs - that each split work among workers

```
#pragma acc parallel
    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j][i] = Anew[j][i];
    }}
    iter++;  }
```

`#pragma acc kernels (!$acc kernels)`

- Similar to parallel, but loops within the kernels region will be independent kernels
- Independent kernels and associated data transfers may be overlapped with other kernels



OpenACC

#pragma acc data (!$acc data)

`#pragma acc data (!$acc data)`

- Defines regions where data may be left on the device
- Useful for reducing PCIe transfers by creating temporary arrays or leaving data on device until needed
- The PGI compiler can automatically migrate data with the `managed` option. Don't use that option if you add the directive!

```
#pragma acc data copy(A, Anew)
while ( error > tol && iter < iter_max ) {
    error=0.f;

#pragma acc parallel
    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            Anew[j][i] = 0.25f * (A[j][i+1] + A[j][i-1]
                                  + A[j-1][i] + A[j+1][i]);
            error = max(error , abs(Anew[j][i] - A[j][i]);
        }
    }
```

`#pragma acc host_data` (`!$acc host_data`)

- Define a region in which host (CPU) arrays will be used, unless specified with use_device()
- The use_device() clause exposes device pointer to the CPU
- Useful for overlapping with CPU computation or calling library routines that expect device memory

OpenACC

Other important directives

`#pragma acc wait` (`!$acc wait`)

- Synchronize with asynchronous activities
- May declare specific conditions or wait on all outstanding requests

`#pragma acc update` (`!$acc update`)

- Update a host or device array within a data region
- Allows updating parts of arrays
- Frequently used around MPI

`#pragma acc loop` (`!$acc loop`)

- Useful for optimizing how the compiler treats specific loops
- May be used to specify the decomposition of the work
- May be used to collapse loop nests for additional parallelism
- May be used to declare kernels as independent of each other

Gang

- Highest level of parallelism, equivalent to CUDA Threadblock. (num_gangs => number of threadblocks in the grid)
- A "gang" loop affects the "CUDA Grid"

Worker

- A member of the gang, equivalent to CUDA thread within a threadblock (num_workers => threadblock size)
- A "worker" loop affects the "CUDA Threadblock"

Vector

- Tightest level of SIMT/SIMD/Vector parallelism, roughly equivalent to CUDA warp or SIMD vector length (vector_length should be a multiple of warp size)
- A "vector" loop affects the SIMT parallelism

async clause

- Declares that control should return to the CPU immediately
- If an integer is passed to async, that integer can be passed as a handle to wait

cache construct

- Cache data in software managed data cache (CUDA shared memory)

declare directive

- Specify that data is to allocated in device memory for the duration of an implicit data region created during the execution of a subprogram

OpenACC

Further optimization

- Use loop collapse() to merge loops and increase parallelism at particular levels
- Improve data movement
- Use compilers existing directives regarding loop optimizations
  - Loop unrolling
  - Loop fusion/fission
  - Loop blocking
- Appropriate data access patterns
  - Memory coalescing (make sure threads run simultaneously, try to access memory that is nearby)
  - bank conflicts (arise because of some specific access pattern of data in shared memory)
  - striding (stride of an array = number of locations in memory between beginnings of successive array elements. An array with stride of exactly the same size as the size of each of its elements is contiguous in memory)

Matrix-matrix multiplication

/* C <- C + A x B */

/* Create a parallel region, fork a team of threads. A, B, C are shared among threads. Iterators i, j, k are private to each thread. */

```
#pragma acc parallel loop
  for (i=0; i<size; i++) {
    for (j=0; j<size; j++) {
      for (k=0; k<size; k++) {
        C[i][j] += A[i][k]*B[k][j];
      }
    }
  }
```

Matrix-matrix multiplication

/* C <- C + A x B */

/* Use kernels to mark a region which contain parallelism and let the compiler determine what can safely be parallelized. */

```
#pragma acc kernels
{
  for (i=0; i<size; i++) {
    for (j=0; j<size; j++) {
      for (k=0; k<size; k++) {
        C[i][j] += A[i][k]*B[k][j];
      }
    }
  }
}
```

PARALLEL LOOP	KERNELS
• Requires analysis by programmer to ensure safe parallelism	• Compiler performs parallel analysis and parallelizes what it believes safe
• Will parallelize what a compiler may miss	• Can cover larger area of code with single directive
• Straightforward path from OpenMP	• Gives compiler additional leeway to optimize.

Both approaches are equally valid and can perform equally well.

OpenACC

Parallel Loop Gang Collapse

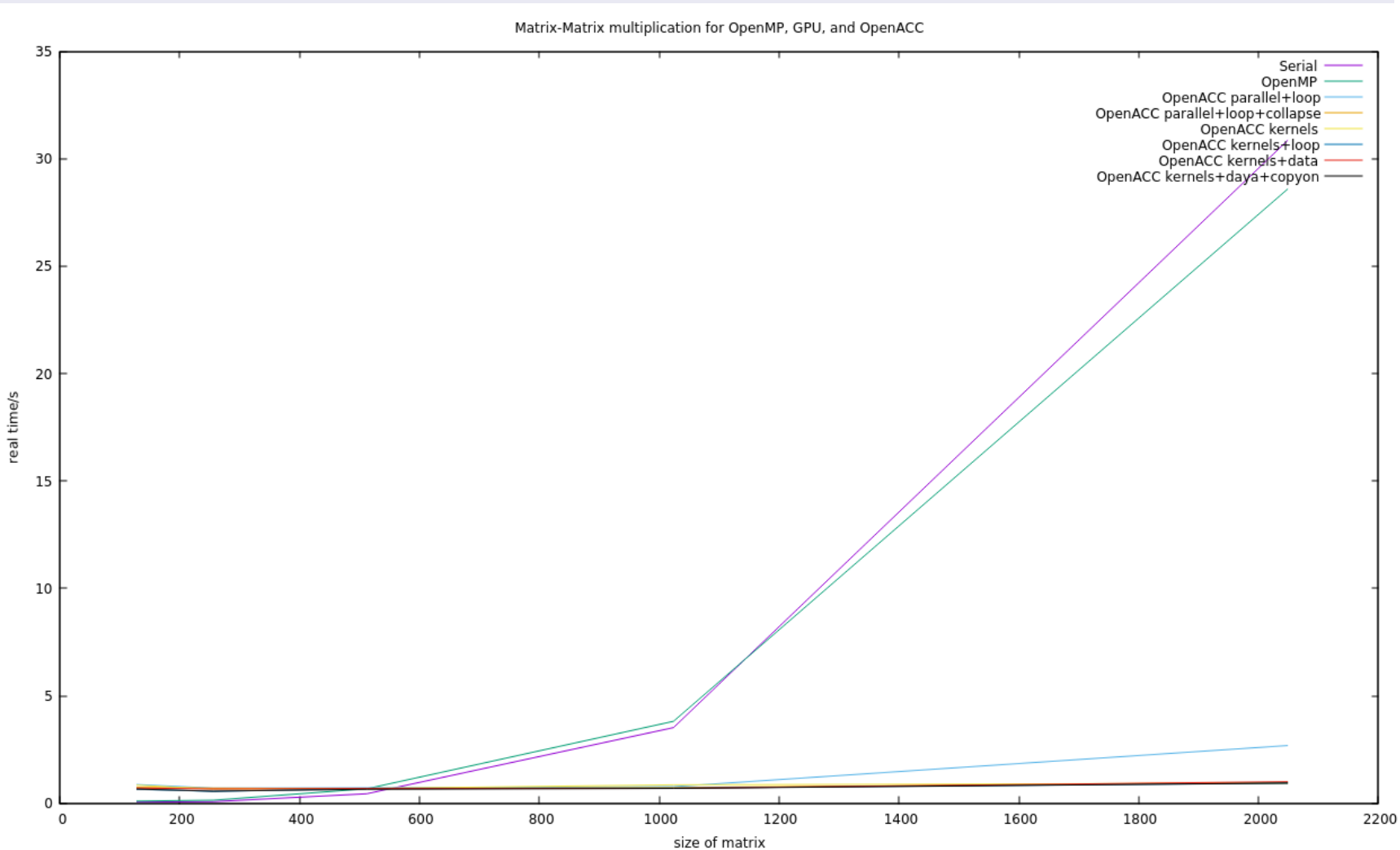
```
#pragma acc parallel loop gang collapse(2)
  for (i=0; i<size; i++) {
    for (j=0; j<size; j++) {
      for (k=0; k<size; k++) {
        C[i][j] += A[i][k]*B[k][j];
      }
    }
  }
```

Table: Matrix-Matrix Multiplication, Comparison of Serial, OpenMP, OpenACC (various directives). Time in s

Size	Serial	OpenMP	parallel+loop	collapse	kernels	kernels+loop	kernels+data	kernels+data+copy
128	0.05	0.102	0.872	0.743	0.841	0.65	0.71	0.66
256	0.076	0.14	0.704	0.702	0.662	0.557	0.668	0.561
512	0.443	0.675	0.72	0.679	0.696	0.682	0.686	0.652
1024	3.523	3.814	0.77	0.730	0.858	0.703	0.693	0.693
2048	30.84	28.574	2.685	0.966	0.936	0.94	1.002	0.933

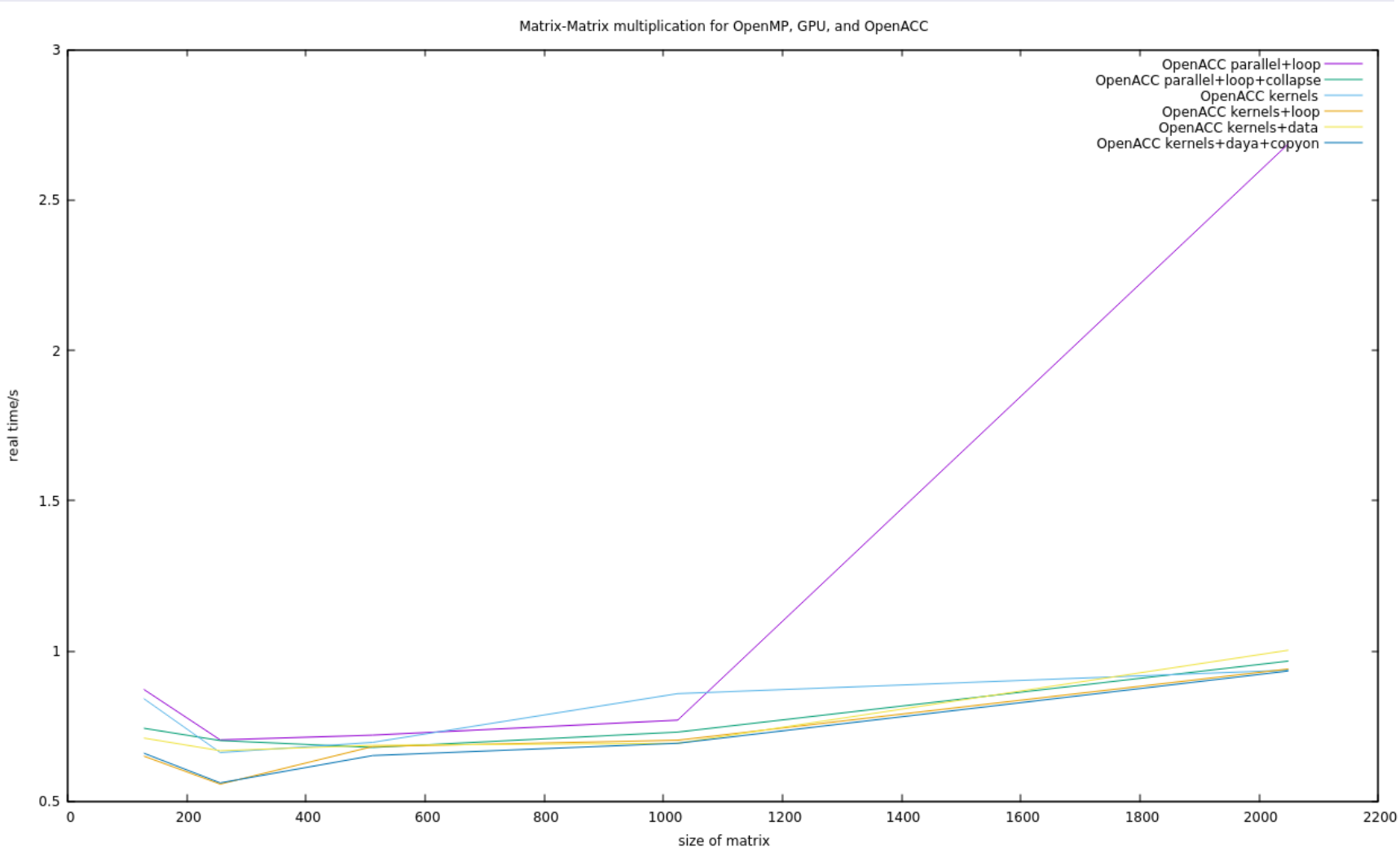
OpenACC

Serial, OpenMP, OpenACC - graphs



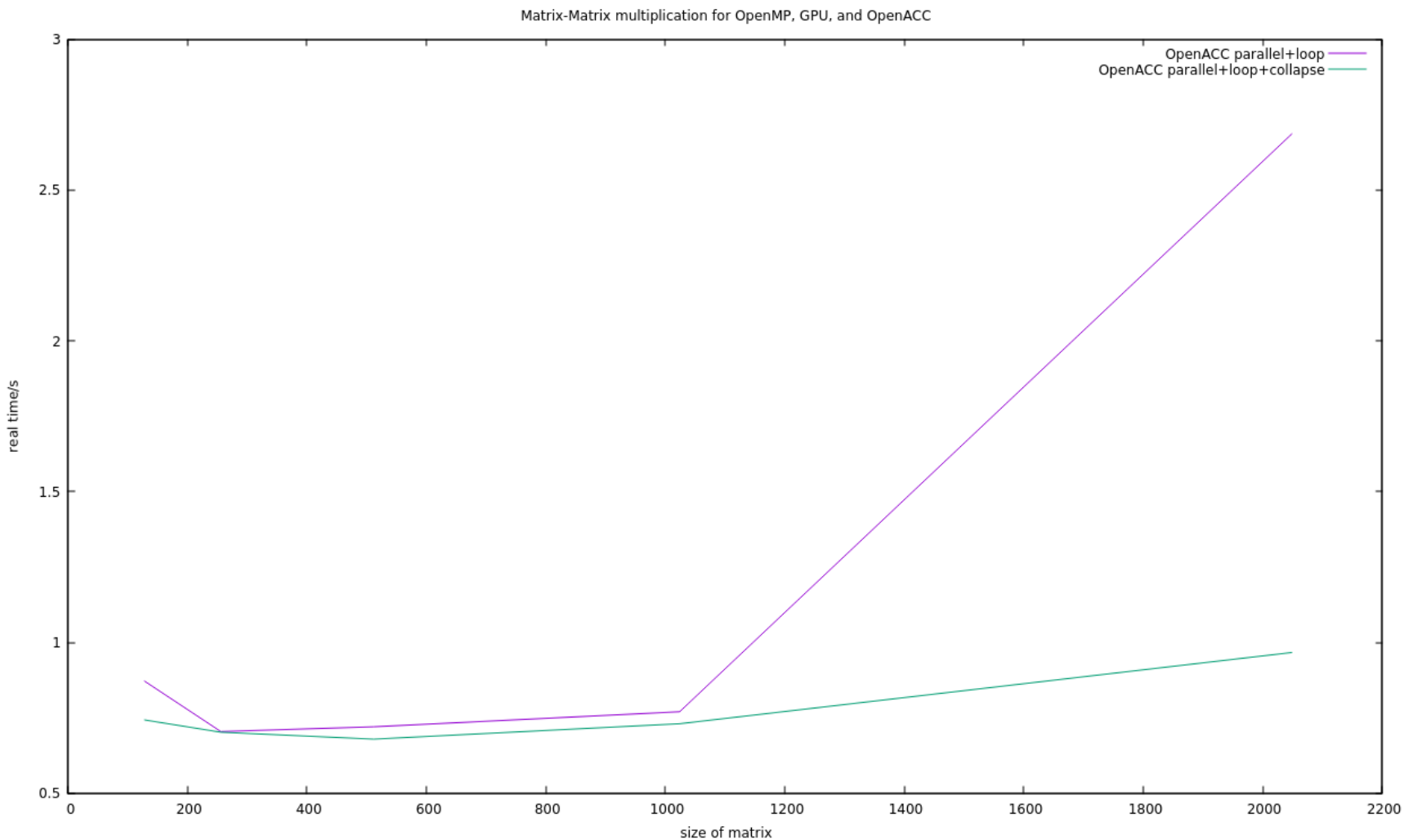
OpenACC

OpenACC graphs



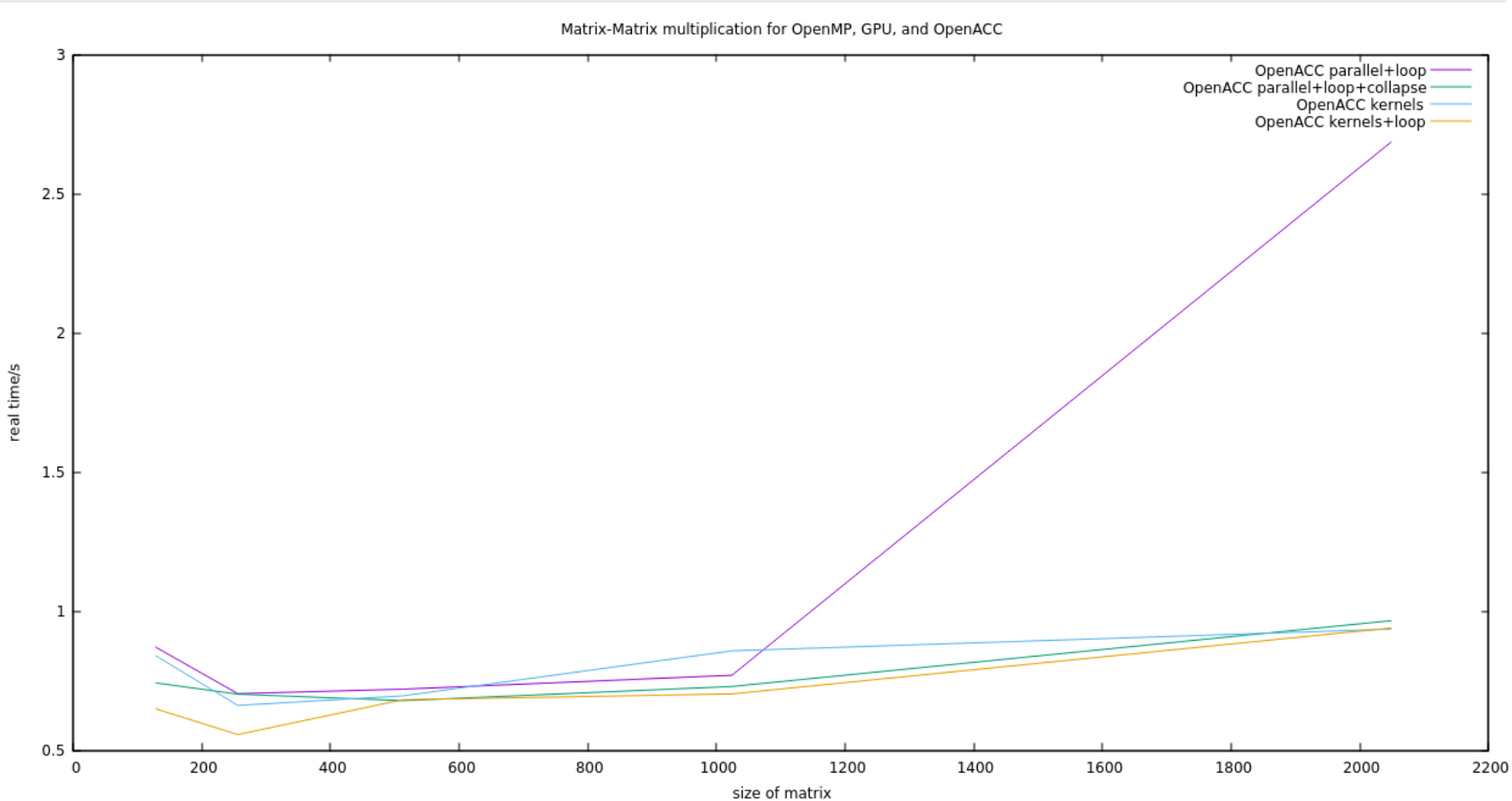
OpenACC

Parallel Loop vs. Kernels



OpenACC

Parallel Loop vs. Kernels



- ml pomkl/2017.10
- **C:** pgcc -acc -ta=tesla -Minfo=accel <filename>.c -o <filename>
- **Fortran:** pgf90 -acc -ta=tesla -Minfo=accel <filename>.f90 -o <filename>

OpenACC

Batch file, example

```
#!/bin/bash
# Change to your own project later!
#SBATCH -A SNIC2017-3-108
#SBATCH --time=00:10:00
#SBATCH --gres=gpu:k80:1

ml purge
ml pomkl/2017.10

time ./openacc-matrix-multiply
```

OpenACC

More information

- https://www.openacc.org/
- https://www.openacc.org/resources
- http://developer.download.nvidia.com/CUDA/training/OpenACC_1_0_intro_jan2012.pdf
- http://on-demand.gputechconf.com/gtc/2015/webinar/Intro-to-OpenACC.pdf
- https://www.olcf.ornl.gov/wp-content/uploads/2013/02/Intro_to_OpenACC-JL.pdf