Future Direction of OpenACC

Cesar Philippidis

Mentor®
A Siemens Business

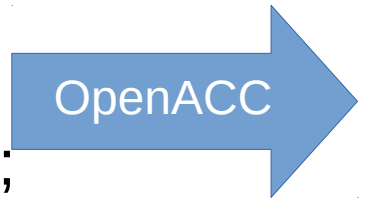
Agenda

- Introduction to OpenACC
- New OpenACC functionality in GCC 9
- Performance Case Study
- Plans for OpenACC 2.6
- Future improvements for OpenACC kernels

What is OpenACC?

- Language extensions to facilitate parallel computing (including GPUs)
- Example: Matrix Multiplication (C[M][N] = A[M][P] * B[P][N])

```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < P; k++)
      c[i][j] += a[i][k] * b[k][j];
```




```
#pragma acc parallel loop gang
for (i = 0; i < M; i++)
  #pragma acc loop worker
  for (j = 0; j < N; j++)
    {
      T t = 0;
    #pragma acc loop vector \
      reduction(+:t)
      for (k = 0; k < P; k++)
        t += a[i][k] * b[k][j];
      c[i][j] = t;
    }
```

New functionality for GCC 9

- Merge changes from openacc-gcc-8-branch (OG8)
  - Many patches!
- Support for OpenACC 2.5
  - Data clause semantics updated to behave more like OpenMP

```
#pragma acc data copy(var)
{
  #pragma acc parallel copy(var)
  ...
}
```

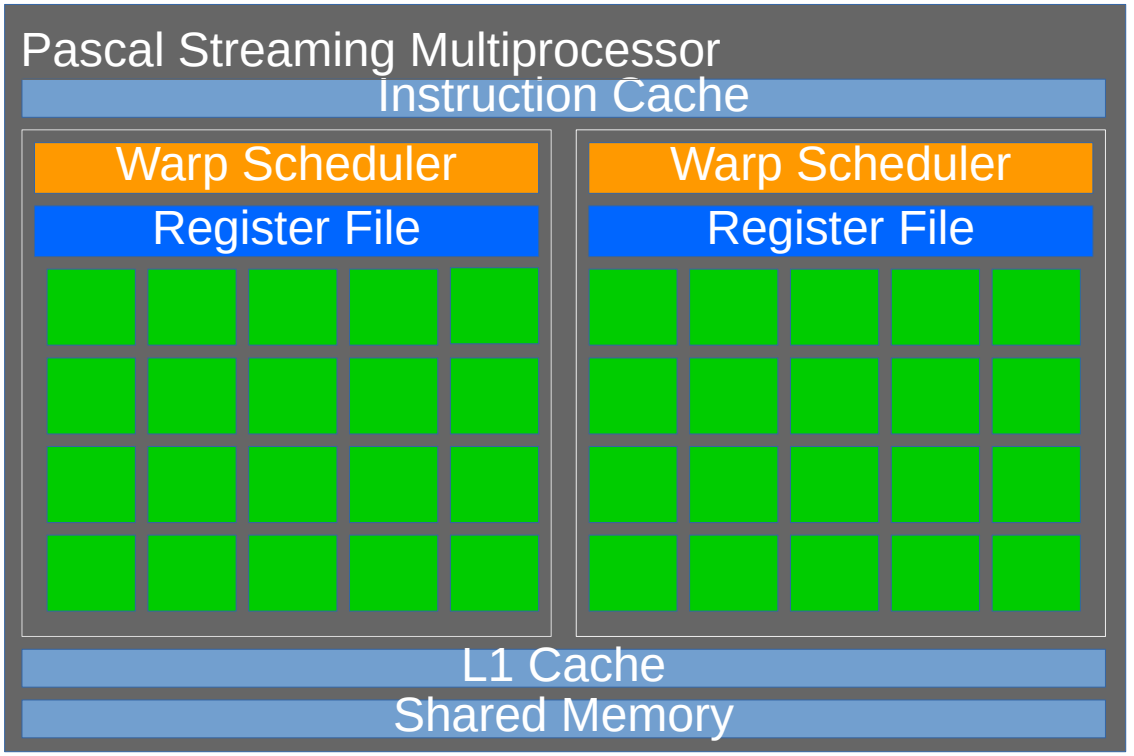
Fortran Changes

- Routines
  - Add support for routines inside Fortran modules
  - Make Fortran FE encode routine parallelism
- Declare
  - Add support for create clause
- Update
  - Add support for derived type objects
  - Not compatible with deep copy

Asynchronous Execution

- OpenACC has a concept of async queues
  - wait(N), async(N)
- Managing multiple async queues in the libgomp plugin is unreliable
  - Async callbacks were broken in early Nvidia drivers
  - Ended up serializing async execution
- Chung-Lin's new async approach makes it a first class citizen in libgomp
  - Introduces new async queue and generic hooks
  - Will be utilized by other devices besides nvptx

Runtime Launch Geometry



Problems:

- Find default launch geometry
- Avoid obscure GPU resource failures

Solution

- Use CUDA Thread Occupancy Calculator

Vector Length

- vector_length clause enable variable-sized vectors

```
#pragma acc parallel loop vector_length(128)
```

- In GCC 8, vector_length = 32 for nvptx
  - Larger vectors can be faked by using worker-vector partitioning

```
#pragma acc parallel num_workers(4)
#pragma acc loop worker vector
```

- Problem
  - Execution model requires two levels of barriers for workers and vectors

```
IF (tid.y != 0) GOTO Lvector_join
IF (tid.x != 0) GOTO Lworker_join
```

NVPTX Vector Length Extension

- GCC 9 will support vector_lengths up to 512
  - Must be a positive integral multiple of 32
- Strategy: Treat large vectors as workers
  - Named bar.sync barriers synchronize vectors
  - Shared-memory used to propagate state
- Extending the vector length exposed existing synchronization bugs with workers (promptly fixed in trunk)
- Limitations
  - Large vector lengths use atomic reduction finalizers
  - oaccdevlow does not utilize larger vector lengths yet

Gang local memory

```
#pragma acc parallel loop gang
for (j = 0; j < n; j++)
{
  int a[n];
  #pragma acc loop worker
  for (i = 0; i < n; i++)
    a[i] = i;
  // do something with ‘a’
}
```

- Solution
  - New “oacc gangprivate” decl attribute inserted during omp-lowering
  - GOACC_EXPAND_ACCEL target hook
- For nvptx, gangprivate variables are placed in shared-memory

Firstprivate reductions

```
#pragma acc parallel loop
  for (i = 0; i < n; i++)
    {
      temp = i;
      #pragma acc loop reduction (+:temp)
      for (j = 0; j < n; j++)
        temp ++;
      a[i] = temp;
    }
```

- Worked with the OpenACC standard committee to allow ‘temp’ to be firstprivate
- DGEMV greatly benefits from this!

C/C++ Dynamic Arrays

- Challenge: multidimensional dynamic arrays may not be contiguous

```
int **a;
float *f[100];
double ***d;

#pragma acc parallel copy (a[0:100][x:y], f[10:20]
[0:30]) copyout(d[x:y][x:y][x:y])
{
 ...
}
```

- Introduces new GOMP_MAP_DYNAMIC_ARRAY map kinds
- Omp-lowering creates and passes array descriptor struct to the runtime
- Doesn’t support “int (*x)[50]” yet

Performance Case Study: LSDalton

- LSDalton
  - Non-trivial quantum chemistry simulator designed for supercomputers
  - Replaced the usage of DGEMM and DGEMV from CUBLAS with lightly modified netlib versions with OpenACC directives
  - Focused on the DEC-RI-MP2 method

Performance Enhancements

- Refined workaround for PTX JIT bug PR83589
  - Forced certain applications to use -ffloat-store
- Use PTX .param space
  - Pass individual parameters to GPU instead of pthread-like struct
  - Not ready for trunk
- Allow inner reduction variables to be firstprivate
- Enable Jakub’s coalesced device data transfer optimization on nvptx

Case Study: LSDalton Cont.

Evolution of openacc-gcc-7-branch performance

Optimization	Execution Time in seconds
Baseline	231.82
PTX JIT Workaround V2	136.92
PTX .param optimization	112.15
Firstprivate Reductions	106.5
Enable Jakub’s coalesced buffer opt.	102.65

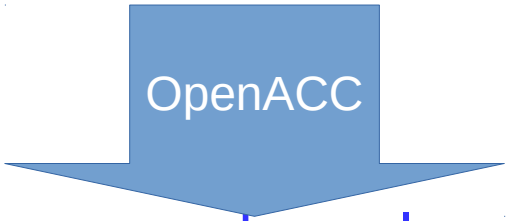
All tests performed on a Nvidia Geforce 1080

GCC 10 and Beyond

- Add support for OpenACC 2.6
  - Attach / detach clauses for deep copy
    - copy(s.scale, s.data[:100])
  - Heterogeneous Memory Management (HMM)
  - Serial construct
- Enhance kernels

Why Kernels Support is Important?

```
for (i = 0; i < M; i++)
  for (j = 0; j < N; j++)
    for (k = 0; k < P; k++)
      c[i][j] += a[i][k] * b[k][j];
```




```
#pragma acc kernels
{
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < P; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```


```
#pragma acc parallel loop gang
for (i = 0; i < M; i++)
  #pragma acc loop worker
  for (j = 0; j < N; j++)
    {
      T t = 0;
    #pragma acc loop vector \
      reduction(+:t)
      for (k = 0; k < P; k++)
        t += a[i][k] * b[k][j];
      c[i][j] = t;
    }
```

Kernels Plan

- Phase I: Expose parallelism inside kernels
  - Teach FE's how to decorate each for/do loop inside kernels region as 'acc loop'
  - Convert acc kernels to acc parallel

```
#pragma acc kernels
{
  for (i = 0; i < M; i++)
    for (j = 0; j < N; j++)
      for (k = 0; k < P; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```


```
#pragma acc parallel
{
  #pragma acc loop auto
  for (i = 0; i < M; i++)
    #pragma acc loop auto
    for (j = 0; j < N; j++)
      #pragma acc loop auto
      for (k = 0; k < P; k++)
        c[i][j] += a[i][k] * b[k][j];
}
```

Kernels Plan Cont.

- Phase II: automatically assign gang, worker, vector partitioning
- Challenges
  - Need strong data dependency analysis
  - Subscripts need to be delinearized
- Idea
  - Add an early gimple pass determine if loops are independent
    - Keep C/Fortran array subscripts in a high-level form
    - Enables gimplifier to more intelligently map variables

Conclusion

- Mentor is committed to the continued support for OpenACC
- GCC 9 will have strong performance on GPUs
- Support for other GPUs coming soon!

Questions