INTRODUCTION TO OPENACC

Nikolay Sakharnykh,
Developer Technology Engineer
OPENACC

The Standard for GPU Directives

- **Simple:** Directives are the easy path to accelerate compute intensive applications
- **Open:** OpenACC is an open GPU directives standard, making GPU programming straightforward and portable across parallel and multi-core processors
- **Portable:** GPU Directives represent parallelism at a high level, allowing portability to a wide range of architectures with the same code.
OPENACC MEMBERS AND PARTNERS
Identify Available Parallelism

Parallelize Loops with OpenACC

Optimize Data Locality

Optimize Loop Performance
CASE STUDY

Jacobi Iteration

- Iteratively converges to correct value (e.g. Temperature), by computing new values at each point from the average of neighboring points.
  - Common, useful algorithm
  - Example: Solve Laplace equation in 2D
  - $\nabla^2 f(x,y) = 0$

$$A_{k+1}(i,j) = \frac{A_k(i-1,j) + A_k(i+1,j) + A_k(i,j-1) + A_k(i,j+1)}{4}$$
JACOBI ITERATION: C CODE

while ( err > tol && iter < iter_max ) {
    err=0.0;

    for( int j = 1; j < n-1; j++) {
        for(int i = 1; i < m-1; i++) {
            Anew[j*m+i] = 0.25 * (A[j*m+i+1] + A[j*m+i-1] +
                          A[(j-1)*m+i] + A[(j+1)*m+i]);
            err = max(err, abs(Anew[j*m+i] - A[j*m+i]));
        }
    }

    for( int j = 1; j < n-1; j++) {
        for( int i = 1; i < m-1; i++ ) {
            A[j*m+i] = Anew[j*m+i];
        }
    }
    iter++;
}
Identify Available Parallelism

Parallelize Loops with OpenACC

Optimize Data Locality

Optimize Loop Performance
IDENTIFY AVAILABLE PARALLELISM

Generating CPU profiling info

- A variety of profiling tools are available:
  - gprof, pgprof, Vampir, Score-p, HPCToolkit, CrayPAT, ...
  - Using the tool of your choice, obtain an application profile to identify hotspots

```
$ pgcc -fast -Minfo=all -Mprof=ccff laplace2d.c

main:
 40, Loop not fused: function call before adjacent loop
    Generated vector sse code for the loop
 57, Generated an alternate version of the loop
    Generated vector sse code for the loop
    Generated 3 prefetch instructions for the loop
 67, Memory copy idiom, loop replaced by call to __c_mcopy8

$ pgcollect ./a.out
$ pgprof -exe ./a.out
```
IDENTIFY AVAILABLE PARALLELISM

PGPROF informs us:

1. A significant amount of time is spent in the loops at line 56/57.

2. The computational intensity (Calculations/Loads&Stores) is high enough to warrant OpenACC or CUDA.

3. How the code is currently optimized.

NOTE: the compiler recognized the swapping loop as data movement and replaced it with a memcpy, but we know it’s expensive too.
while ( err > tol && iter < iter_max ) {
    err=0.0;

    for( int j = 1; j < n-1; j++ ) {
        for(int i = 1; i < m-1; i++) {
            Anew[j*m+i] = 0.25 * (A[j*m+i+1] + A[j*m+i-1] +
                A[(j-1)*m+i] + A[(j+1)*m+i]);

            err = max(err, abs(Anew[j*m+i] - A[j*m+i]));
        }
    }

    for( int j = 1; j < n-1; j++ ) {
        for( int i = 1; i < m-1; i++ ) {
            A[j*m+i] = Anew[j*m+i];
        }
    }
    iter++;
}
OPENACC DIRECTIVE SYNTAX

"C/C++"

```
#pragma acc directive [clause [,] clause] ...
```
...often followed by a structured code block

"Fortran"

```
!$acc directive [clause [,] clause] ...
```
...often paired with a matching end directive surrounding a structured code block:

```
!$acc end directive
```
The kernels construct expresses that a region may contain parallelism and the compiler determines what can safely be parallelized.

```c
#pragma acc kernels
{
    for(int i=0; i<N; i++)
    {
        x[i] = 1.0;
        y[i] = 2.0;
    }
    for(int i=0; i<N; i++)
    {
        y[i] = a*x[i] + y[i];
    }
}
```

The compiler identifies 2 parallel loops and generates 2 kernels.
while (err > tol && iter < iter_max) {
    err=0.0;

    #pragma acc kernels
    {
        for( int j = 1; j < n-1; j++) {
            for(int i = 1; i < m-1; i++) {

                Anew[j*m+i] = 0.25 * (A[j*m+i+1] + A[j*m+i-1] + A[(j-1)*m+i] + A[(j+1)*m+i]);

                err = max(err, abs(Anew[j*m+i] - A[j*m+i]));
            }
        }

        for( int j = 1; j < n-1; j++) {
            for( int i = 1; i < m-1; i++ ) {
                A[j*m+i] = Anew[j*m+i];
            }
        }
    }

    iter++;
}
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:

85, Accelerator restriction: size of the GPU copy of Anew,A is unknown
 Loop carried dependence of Anew-> prevents parallelization
 Loop carried dependence of Anew-> prevents vectorization
 Loop carried backward dependence of Anew-> prevents vectorization
 Generating copyin(A[:])
 Generating copyout(Anew[:])

86, Loop is parallelizable
 Accelerator kernel generated
 Generating Tesla code
 86, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */

91, Max reduction generated for error

95, Accelerator restriction: size of the GPU copy of A,Anew is unknown
 Loop carried dependence of A-> prevents parallelization
 Loop carried backward dependence of A-> prevents vectorization
 Generating copyout(A[:])
 Generating copyin(Anew[:])

96, Loop is parallelizable
 Accelerator kernel generated
 Generating Tesla code
 96, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
$ pgcc -fast -acc -ta=tesla -Minfo=all laplace2d.c
main:

85, Accelerator restriction: size of the GPU copy of Anew,A is unknown
  Loop carried dependence of Anew-> prevents parallelization
  Loop carried dependence of Anew-> prevents vectorization
  Loop carried backward dependence of Anew-> prevents vectorization
  Generating copyin(A[:])
  Generating copyout(Anew[:])

86, Loop is parallelizable
  Accelerator kernel generated
  Generating Tesla code

86, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */

91, Max reduction generated for error

95, Accelerator restriction: size of the GPU copy of A,Anew is unknown
  Loop carried dependence of A-> prevents parallelization
  Loop carried backward dependence of A-> prevents vectorization
  Generating copyout(A[:])
  Generating copyin(Anew[:])

96, Loop is parallelizable
  Accelerator kernel generated
  Generating Tesla code

96, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
-ta=tesla:managed offloads data management to CUDA driver by using Unified Memory

```c
a = (float*)malloc(sizeof(float) * n);
for(int i=0; i<N; i++) a[i] = i;
#pragma acc kernels
{
    for(int i=0; i<N; i++) a[i] *= 2;
}
printf("%f %f %f\n", a[0],a[1],a[2]);
```

- `a[]` allocated on heap
- accessed from **CPU**
- accessed from **GPU**, compiler doesn’t have to insert explicit data copy
- accessed from **CPU**
$ pgcc -fast -acc -ta=tesla:managed -Minfo=all laplace2d.c
main:
83, Generating copyout(Anew[:])
   Generating copy(A[:])
85, Loop carried dependence of Anew-> prevents parallelization
   Loop carried dependence of Anew-> prevents vectorization
   Loop carried backward dependence of Anew-> prevents vectorization
86, Loop is parallelizable
   Accelerator kernel generated
   Generating Tesla code
86, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
91, Max reduction generated for error
95, Loop carried dependence of A-> prevents parallelization
   Loop carried backward dependence of A-> prevents vectorization
96, Loop is parallelizable
   Accelerator kernel generated
   Generating Tesla code
96, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
$ pgcc -fast -acc -ta=tesla:managed -Minfo=all laplace2d.c
main:
  83, Generating copyout(Anew[:])
  Generating copy(A[:])
  85, Loop carried dependence of Anew-> prevents parallelization
      Loop carried dependence of Anew-> prevents vectorization
      Loop carried backward dependence of Anew-> prevents vectorization
  86, Loop is parallelizable
      Accelerator kernel generated
      Generating Tesla code
      86, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
  91, Max reduction generated for error
  95, Loop carried dependence of A-> prevents parallelization
      Loop carried backward dependence of A-> prevents vectorization
  96, Loop is parallelizable
      Accelerator kernel generated
      Generating Tesla code
      96, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
OPENACC PARALLEL LOOP DIRECTIVE

- **parallel** - Programmer identifies a block of code containing parallelism. Compiler generates a **kernel**.
- **loop** - Programmer identifies a loop that can be parallelized within the kernel.
- **NOTE**: parallel & loop are often placed together

```c
#pragma acc parallel loop
for(int i=0; i<N; i++)
{
    y[i] = a*x[i]+y[i];
}
```

**Kernel:**
A function that runs in parallel on the GPU
OPENACC INDEPENDENT CLAUSE

- Specifies that loop iterations are data independent. This overrides any compiler dependency analysis. This is implied for parallel loop.

```c
#pragma acc kernels
{
  #pragma acc loop independent
  for(int i=0; i<N; i++)
  {
    a[i] = 0.0;
    b[i] = 1.0;
    c[i] = 2.0;
  }
  #pragma acc loop independent
  for(int i=0; i<N; i++)
  {
    a(i) = b(i) + c(i)
  }
}
```

Informs the compiler that both loops are safe to parallelize so it will generate both kernels.
while ( err > tol && iter < iter_max ) {
  err=0.0;

#pragma acc kernels
{
  #pragma acc loop independent
  for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {

      Anew[j*m+i] = 0.25 * (A[j*m+i+1] + A[j*m+i-1] + A[(j-1)*m+i] + A[(j+1)*m+i]);

      err = max(err, abs(Anew[j*m+i] - A[j*m+i]));
    }
  }

#pragma acc loop independent
  for( int j = 1; j < n-1; j++ ) {
    for( int i = 1; i < m-1; i++ ) {
      A[j*m+i] = Anew[j*m+i];
    }
  }
}

iter++;
}
$ pgcc -fast -acc -ta=tesla:managed -Minfo=all laplace2d.c

main:

83, Generating copyout(Anew[:])
    Generating copy(A[:])
86, Loop is parallelizable
87, Loop is parallelizable
    Accelerator kernel generated
    Generating Tesla code
    86, #pragma acc loop gang /* blockIdx.y */
    87, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
    92, Max reduction generated for error
97, Loop is parallelizable
98, Loop is parallelizable
    Accelerator kernel generated
    Generating Tesla code
    97, #pragma acc loop gang /* blockIdx.y */
    98, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
PERFORMANCE RESULTS

Speed-up (Higher is Better)

Intel Xeon E5-2698 v3 @ 2.30GHz (Haswell) vs. NVIDIA Tesla K40 (Kepler)

Socket(Socket: 6.86X

<table>
<thead>
<tr>
<th>Core Configuration</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Thread</td>
<td>1.0</td>
</tr>
<tr>
<td>2 Threads</td>
<td>2.0</td>
</tr>
<tr>
<td>4 Threads</td>
<td>3.5</td>
</tr>
<tr>
<td>6 Threads</td>
<td>4.3</td>
</tr>
<tr>
<td>8 Threads</td>
<td>4.4</td>
</tr>
<tr>
<td>OpenACC</td>
<td>30.2</td>
</tr>
</tbody>
</table>
Identify Available Parallelism

Optimize Loop Performance

Parallelize Loops with OpenACC

Optimize Data Locality
Any tool that supports CUDA can likewise obtain performance information about OpenACC.

NVIDIA Visual Profiler (nvvp) comes with the CUDA Toolkit, so it will be available on any machine with CUDA installed.
OPTIMIZE PERFORMANCE

Data locality

- Use explicit data regions
  - The `data` construct defines a region of code in which GPU arrays remain on the GPU and are shared among all kernels in that region.

```c
#pragma acc data copyin(a[0:size]), copyout(b[s/4:3*s/4])
```

- Unstructured data regions
  - Used to define data regions when scoping doesn’t allow the use of normal data regions (e.g. constructor/destructor of a class)

```c
#pragma acc enter data copyin(a)
...
#pragma acc exit data delete(a)
```
OPTIMIZE PERFORMANCE

Loop performance

- Fine control of loop parallelism
  - `gang`, `worker`, and `vector` can be added to a loop clause
  - Control the size using `num_gangs(n)`, `num_workers(n)`, `vector_length(n)`

```c
#pragma acc kernels loop gang
for (int i = 0; i < n; ++i)
#pragma acc loop vector(128)
for (int j = 0; j < n; ++j)
...
```
WRAP UP

- Identify Available Parallelism
  - What important parts of the code have available parallelism?

- Parallelize Loops
  - Express as much parallelism as possible and ensure you still get correct results.

- Optimize Data Locality
  - Use unified memory if possible, then hand-tune migration with data directives.

- Optimize Loop Performance
  - Don’t try to optimize a kernel that runs in a few us or ms until you’ve eliminated the excess data motion that is taking many seconds.
OPENACC RESOURCES

- OpenACC toolkit

- GTC on-demand and webinars
  - http://on-demand-gtc.gputechconf.com
  - http://www.gputechconf.com/gtc-webinars

- Parallel Forall Blog
  - http://devblogs.nvidia.com/parallelforall

- Self-paced labs
  - http://nvlabs.qwiklab.com