GPU Computing with OpenACC Directives

Alexey Romanenko

Based on Jeff Larkin's PPTs

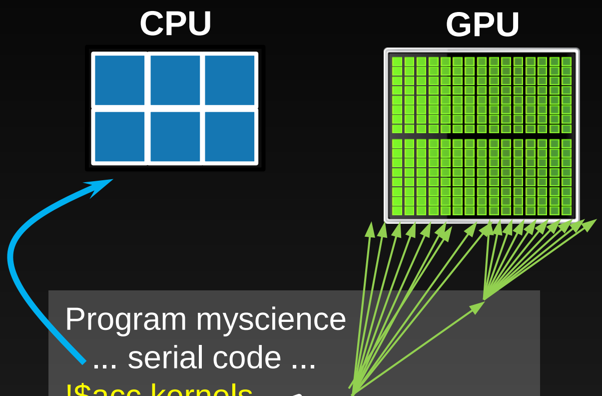
NVIDIA

3 Ways to Accelerate Applications

Applications

Libraries	OpenACC Directives	Programming Languages
"Drop-in" Acceleration	Easily Accelerate Applications	Maximum Flexibility

OpenACC Directives




```
Program myscience
  ... serial code ...
!$acc kernels
  do k = 1,n1
    do i = 1,n2
        ... parallel code ...
    enddo
   enddo
!$acc end kernels
  ...
End Program myscience
```

OpenACC Compiler Hint

Your original Fortran or C code

Simple Compiler hints

Compiler Parallelizes code

Works on many-core GPUs & multicore CPUs

OpenACC
Open Programming Standard for Parallel Computing

“OpenACC will enable programmers to easily develop portable applications that maximize the performance and power efficiency benefits of the hybrid CPU/GPU architecture of Titan.”

--Buddy Bland, Titan Project Director, Oak Ridge National Lab

“OpenACC is a technically impressive initiative brought together by members of the OpenMP Working Group on Accelerators, as well as many others. We look forward to releasing a version of this proposal in the next release of OpenMP.”

--Michael Wong, CEO OpenMP Directives Board

OpenACC Standard

NVIDIA CRAY THE SUPERCOMPUTER COMPANY PGI CAPS

OpenACC

The Standard for GPU Directives

- **Easy:** 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
- **Powerful:** GPU Directives allow complete access to the massive parallel power of a GPU

2 Basic Steps to Get Started

- **Step 1: Annotate source code with directives:**

```
!$acc data copy(util1,util2,util3) copyin(ip,scp2,scp2i)
  !$acc parallel loop
  …
  !$acc end parallel
!$acc end data
```

- **Step 2: Compile & run:**

```
pgf90 -ta=nvidia -Minfo=accel file.f
```

OpenACC Directives Example

```
iter=0
do while ( err > tol .and. iter < iter_max )

 iter = iter +1
 err=0._fp_kind



   do j=1,m
    do i=1,n
     Anew(i,j) = .25_fp_kind *( A(i+1,j  ) + A(i-1,j  ) &
                               +A(i  ,j-1) + A(i  ,j+1))
     err = max( err, Anew(i,j)-A(i,j))
    end do
   end do

   A = Anew

   IF(mod(iter,100)==0 .or. iter == 1)    print *, iter, err
end do
```

OpenACC Directives Example

```
!$acc data copy(A) create(Anew)
 iter=0
 do while ( err > tol .and. iter < iter_max )

  iter = iter +1
  err=0._fp_kind

!$acc kernels
   do j=1,m
    do i=1,n
     Anew(i,j) = .25_fp_kind *( A(i+1,j  ) + A(i-1,j  ) &
                              +A(i  ,j-1) + A(i  ,j+1))
     err = max( err, Anew(i,j)-A(i,j))
    end do
   end do

   A = Anew
!$acc end kernels
   IF(mod(iter,100)==0 .or. iter == 1)    print *, iter, err
 end do
!$acc end data
```

Copy arrays into GPU memory within data region

Parallelize code inside region

Close off parallel region

Close off data region, copy data back

OpenACC Directives

- Parallel execution
  - `parallel`, `kernels`, `loop`
- Data management
  - `data`, `enter data`, `exit data`, `update`
- Other
  - `routine`
  - `atomic`
  - `host_data`
  - `wait`

OpenACC "parallel" Directive

parallel - Programmer identifies a block of code containing parallelism. Compiler generates a kernel.

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

acc parallel [clauses]

clauses:

- async
- if
- reduction
- num_gangs
- vector_length
- device_type
- …

OpenACC “kernels” Directive

The kernels construct expresses that a region *may contain parallelism* and *the compiler determines* what can safely be parallelized.

```
#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];
}
}
```

kernel 1

kernel 2

acc kernels [clauses]
clauses:

- async
- if
- device_type
- …

OpenACC “parallel” vs. “kernels”

PARALLEL LOOP

- Requires analysis by programmer to ensure safe parallelism
- Will parallelize what a compiler may miss
- Straightforward path from OpenMP

KERNELS

- Compiler performs parallel analysis and parallelizes what it believes safe
- Can cover larger area of code with single directive
- Gives compiler additional leeway to optimize.

OpenACC "async" and "wait"

async(n): launches work asynchronously in queue *n*

wait(n): blocks host until all operations in queue *n* have completed

Can significantly reduce launch latency, enables pipelining and concurrent operations

```
!$acc parallel loop async(1)
! do loop here
!$acc end parallel
 call do_something_on_cpu()
!$acc wait(1)
```

OpenACC "loop" Directive

The loop directive describes what type of parallelism to use to execute the loop

Clauses:

- independent
- collapse (n)
- private (var-list)
- reduction (operator:var-list)
- gang [(int-expresion)]
- vector [(int-expresion)]
- …

OpenACC “loop” directive: private & reduction

- The private and reduction clauses are not optimization clauses, they may be required for correctness.
- private – A copy of the variable is made for each loop iteration
- reduction - A reduction is performed on the listed variables.
  - Supports +, *, max, min, and various logical operations

```
!$acc loop private(tmp) reduction(max:err)
do I=1,M
     tmp = a(I-1) + 2.0*a(I)…
     err = max(err,tmp)
enddo
```

OpenACC “loop” directive: gang & vector

- The gang clause specifies that the iterations of the associated
- loop or loops are to be executed in parallel.
- The vector clause specifies that the iterations of the associated loop or loops are to be executed in vector or SIMD mode.

```
!$acc loop gang vector(16)
do I=2,M-1
!$acc loop gang vector(16)
   do J=2,N-1
      out(J,I) = coef*(a(J-1,I-1)+a(J,I-1)…
   enddo
enddo
```

Managed Memory

- Works for
  - NVIDIA Kepler GPU and newer
  - 64-bit Linux OS
  - dynamically-allocated memory
- Compiler's flag
  - pgfortran -ta=nvidia:managed

OpenACC "data" Directive

`copy ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region and copies data to the host when exiting region.
`copyin ( list )`	Allocates memory on GPU and copies data from host to GPU when entering region.
`copyout ( list )`	Allocates memory on GPU and copies data to the host when exiting region.
`create ( list )`	Allocates memory on GPU but does not copy.
`present ( list )`	Data is already present on GPU from another containing data region.

and `present_or_copy[in|out]`, `present_or_create`, `deviceptr`.

OpenACC Directives Example

```
!$acc data copy(A) create(Anew)
 iter=0
 do while ( err > tol .and. iter < iter_max )

  iter = iter +1
  err=0._fp_kind
!$acc kernels
   do j=1,m
    do i=1,n
     Anew(i,j) = .25_fp_kind *( A(i+1,j  ) + A(i-1,j  ) &
                               +A(i  ,j-1) + A(i  ,j+1))
     err = max( err, Anew(i,j)-A(i,j))
    end do
   end do

   A = Anew
!$acc end kernels
   IF(mod(iter,100)==0 .or. iter == 1)    print *, iter, err
 end do
!$acc end data
```

Copy array “A” into GPU memory within data region, create array “Anew”

OpenACC "enter data" & "exit data"

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

`enter data` Defines the start of an unstructured data lifetime

clauses: `copyin(list)`, `create(list)`, `present_or_copyin(list)`, `present_or_create(list)`

`exit data` Defines the end of an unstructured data lifetime

clauses: `copyout(list)`, `delete(list)`

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

Array Shaping

- Compiler sometimes cannot determine size of arrays
- Must specify explicitly using data clauses and array “shape”
- C/C++
  - #pragma acc data copyin(a[0:size]), copyout(b[s/4:3*s/4])
- Fortran
  - !$acc data copyin(a(1:end)), copyout(b(s/4:3*s/4))

OpenACC "update" Directive

Programmer specifies an array (or part of an array) that should be refreshed within a data region.

```
do_something_on_device()
!$acc update self(a)

do_something_on_host()
!$acc update device(a)
```

Copy "a" from GPU to CPU

Copy "a" from CPU to GPU

OpenACC “routine” Directive

Specifies that the compiler should generate a device copy of the function/subroutine and what type of parallelism the routine contains.

Clauses:

- gang/worker/vector/seq
  - Specifies the level of parallelism contained in the routine.
- bind
  - Specifies an optional name for the routine, also supplied at call-site
- no_host
  - The routine will only be used on the device
- device_type
  - Specialize this routine for a particular device type.

OpenACC "routine" Directive

```
// mandelbrot.h
#pragma acc routine seq
unsigned char mandelbrot(int Px, int Py);

// Used in main()
#pragma acc parallel loop
for(int y=0;y<HEIGHT;y++) {
  for(int x=0;x<WIDTH;x++) {
    image[y*WIDTH+x]=mandelbrot(x,y);
  }
}
```

- At function source:
  - Function needs to be built for the GPU.
  - It will be called by each thread (sequentially)
- At call the compiler needs to know:
  - Function will be available on the GPU
  - It is a sequential routine

OpenACC “atomic” Directive

atomic: subsequent block of code is performed atomically with respect to other threads on the accelerator

Clauses: read, write, update, capture

```
#pragma acc parallel loop
for(int i=0; i<N; i++) {
  #pragma acc atomic update
  a[i%100]++;
}
```

Interoperability

OpenACC plays well with others.

- Add CUDA, OpenCL, or accelerated libraries to an OpenACC application
- Add OpenACC to an existing accelerated application
- Share data between OpenACC and CUDA

OpenACC "host_data" directive

Exposes the *device* address of particular objects to the *host* code.

```
#pragma acc data copy(x,y)
{
// x and y are host pointers
#pragma acc host_data use_device(x,y)
{
 // x and y are device pointers
}
// x and y are host pointers
}
```

X and Y are device pointers here

OpenACC "host_data" Example

```
program main
  integer, parameter :: N = 2**20
  real, dimension(N) :: X, Y
  real               :: A = 2.0

  !$acc data
  ! Initialize X and Y
  ...

  !$acc host_data use_device(x,y)
  call saxpy(n, a, x, y)
  !$acc end host_data
  !$acc end data

end program
```

```
__global__
void saxpy_kernel(int n, float a,
                  float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

void saxpy(int n, float a, float *dx, float *dy)
{
  // Launch CUDA Kernel
  saxpy_kernel<<<4096,256>>>(N, 2.0, dx, dy);
}
```

- It's possible to interoperate from C/C++ or Fortran.
- OpenACC manages the data and passes device pointers to CUDA.

- CUDA kernel launch wrapped in function expecting device arrays.
- Kernel is launch with arrays passed from OpenACC in main.

CUBLAS Librry & OpenACC

OpenACC can interface with existing GPU-optimized libraries (from C/C++ or Fortran).

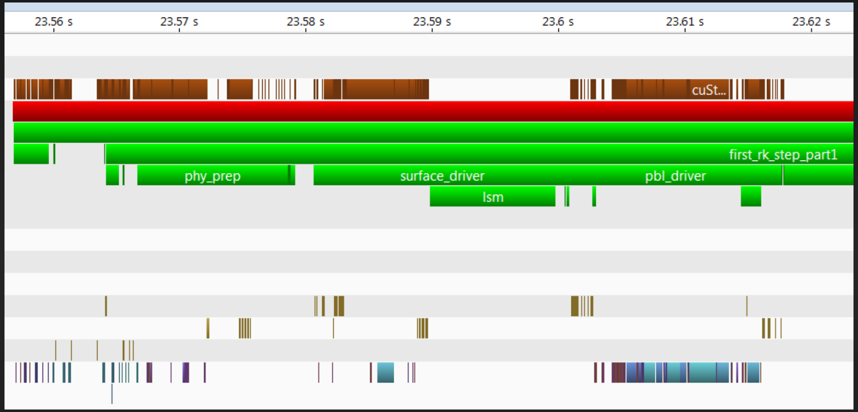
This includes…

- CUBLAS
- Libsci_acc
- CUFFT
- MAGMA
- CULA
- Thrust
- …

```
int N = 1<<20;
float *x, *y
// Allocate & Initialize X & Y
...

cublasInit();

#pragma acc data copyin(x[0:N]) copy(y[0:N])
{
  #pragma acc host_data use_device(x,y)
  {
    // Perform SAXPY on 1M elements
    cublasSaxpy(N, 2.0, x, 1, y, 1);
  }
}

cublasShutdown();
```

Profiling

- PGI_ACC_TIME=1
- nvprof, nvvp
  - mpirun -np <n> nvprof -o name.%p.nvprof <program>
  - mpirun -np <n> nvprof -o name.%q{OMPI_COMM_WORLD_RANK}.nvprof <program>
- Use NVTX library
  http://devblogs.nvidia.com/parallelforall/customize-cuda-fortran-profiling-nvtx/



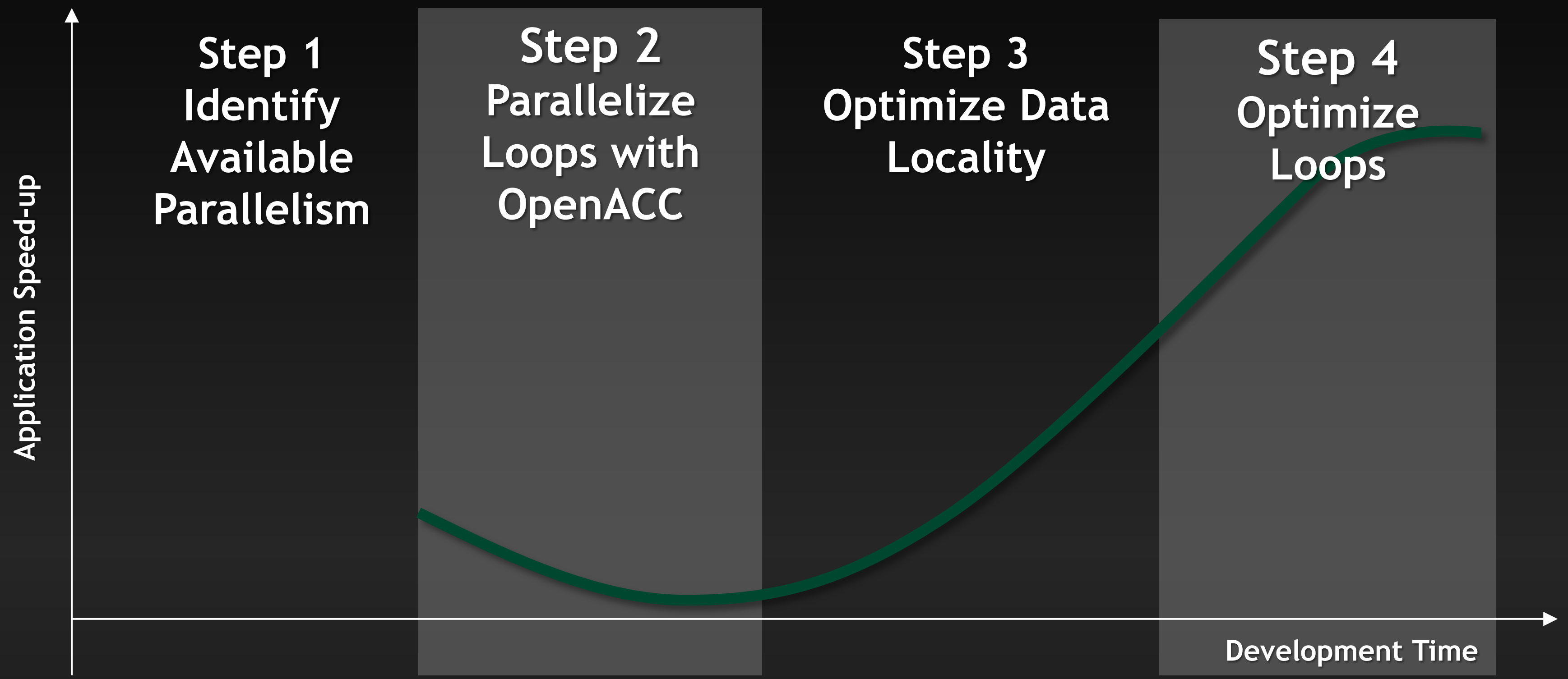
Debugging

- PGI_ACC_NOTIFY={bit mask}
  - 1 - kernels launch, 2 - data transfers, 4 - sync operations, 8 - region entry/exit, 16 - data allocation/free
- PGI_ACC_DEBUG=1
- PGI_ACC_SYNCHRONOUS=1
- Use “if” clause and “update” directives

Process of Adaptation

- 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.
  - Because the compiler must be cautious about data movement, the code will generally slow down.
- Optimize Data Locality
  - The programmer will always know better than the compiler what data movement is unnecessary.
- 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.

Typical Porting Experience with OpenACC Directives



Misc Advices

Write Parallelizable loops

- Avoid pointer arithmetic
- Write countable loops
- Write rectangular loops

```
for(int i=0;i<N;i++)
  for(int j=i;j<N;j++)
    sum+=A[i][j];
```

```
for(int i=0;i<N;i++)
  for(int j=0;j<N;j++)
    if(j>=i)
      sum+=A[i][j];
```

```
bool found=false;
while(!found && i<N){
  if(a[i]==val){
    found=true; loc=i;
  }
  i++;
}
```

```
bool found=false;
for(int i=0;i<N;i++){
  if(a[i]==val && found == false){
    found=true
    loc=i;
  }
}
```

C99: “restrict” keyword

- Declaration of intent given by the programmer to the compiler

  Applied to a pointer, e.g.

  float *restrict ptr

  Meaning: “for the lifetime of ptr, only it or a value directly derived from it (such as ptr + 1) will be used to access the object to which it points”

- Parallelizing compilers often require restrict to determine independence
  - Otherwise the compiler can’t parallelize loops that access ptr

```
~~float restrict *ptr~~
float *restrict ptr
```

http://en.wikipedia.org/wiki/Restrict

OpenACC: "collapse" clause

collapse(n): Transform the following n tightly nested loops into one, flattened loop.

Useful when individual loops lack sufficient parallelism or more than 3 loops are nested (gang/worker/vector)

```
#pragma acc parallel
#pragma acc loop collapse(2)
for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    ...
```

```
#pragma acc parallel
#pragma acc loop
for(int ij=0; ij<N*N; ij++)
    ...
```

Kernel Fusion

- Kernel calls are expensive
  - Each call can take over 10us in order to launch
  - It is often a good idea to combine loops of same trip counts containing very few lines of code
- Kernel Fusion (i.e. Loop fusion)
  - Join nearby kernels into a single kernel

```
#pragma acc parallel loop
for (int i = 0; i < n; ++i)
    a[i]=0;
#pragma acc parallel loop
for (int i = 0; i < n; ++i)
    b[i]=0;
```

```
#pragma acc parallel loop
  for (int i = 0; i < n; ++i) {
    a[i]=0;
    b[i]=0;
}
```

Loop Fusion

- Loops that are exceptionally long may result in kernels that are resource-bound, resulting in low GPU occupancy.
- This is particularly true for outer parallel loops containing nested loops
- Caution: This may introduce temporaries.

```
#pragma acc parallel loop
for (int j = 0; j < m; ++j ) {
  for (int i = 0; i < n; ++i) {
    a[i]=0;
  }
  for (int i = 0; i < n; ++i) {
    b[i]=0;
  }
}
```

```
#pragma acc parallel loop
for (int j = 0; j < m; ++j )
  for (int i = 0; i < n; ++i) {
    a[i]=0;
  }
#pragma acc parallel loop
for (int j = 0; j < m; ++j )
  for (int i = 0; i < n; ++i) {
    b[i]=0;
  }
```

OpenACC Debugging

- Most OpenACC directives accept an `if(condition)` clause

```
#pragma acc update self(A) if(debug)
#pragma acc parallel loop if(!debug)
[…]
#pragma acc update device(A) if(debug)
```

- Use `default(none)` to force explicit data directives

```
#pragma acc data copy(…) create(…) default(none)
```

Directives: Easy & Powerful

Real-Time Object Detection

Global Manufacturer of Navigation Systems

5x in 40 Hours

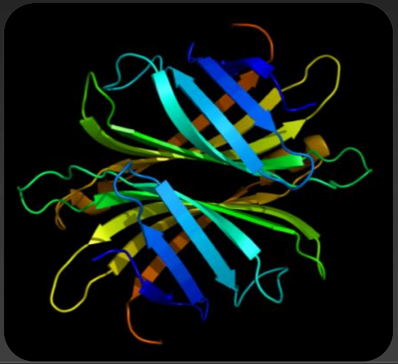
Valuation of Stock Portfolios using Monte Carlo

Global Technology Consulting Company

2x in 4 Hours

Interaction of Solvents and Biomolecules

University of Texas at San Antonio



5x in 8 Hours

"Optimizing code with directives is quite easy, especially compared to CPU threads or writing CUDA kernels. The most important thing is avoiding restructuring of existing code for production applications."

-- Developer at the Global Manufacturer of Navigation Systems

Start Now with OpenACC Directives

- Get free trial license to PGI Accelerator
  - http://www.nvidia.com/gpudirectives
- Sign up for a free online course
  - https://developer.nvidia.com/openacc-course
- Get your free OpenACC Toolkit
  - http://www.nvidia.com/object/openacc-toolkit.html

Thank you