

INTRODUCTION TO OPENACC

Lecture 3: Advanced, November 9, 2016





Course Objective:

Enable *you* to accelerate your applications
with OpenACC.

Course Syllabus

Oct 26: Analyzing and Parallelizing with OpenACC

Nov 2: OpenACC Optimizations

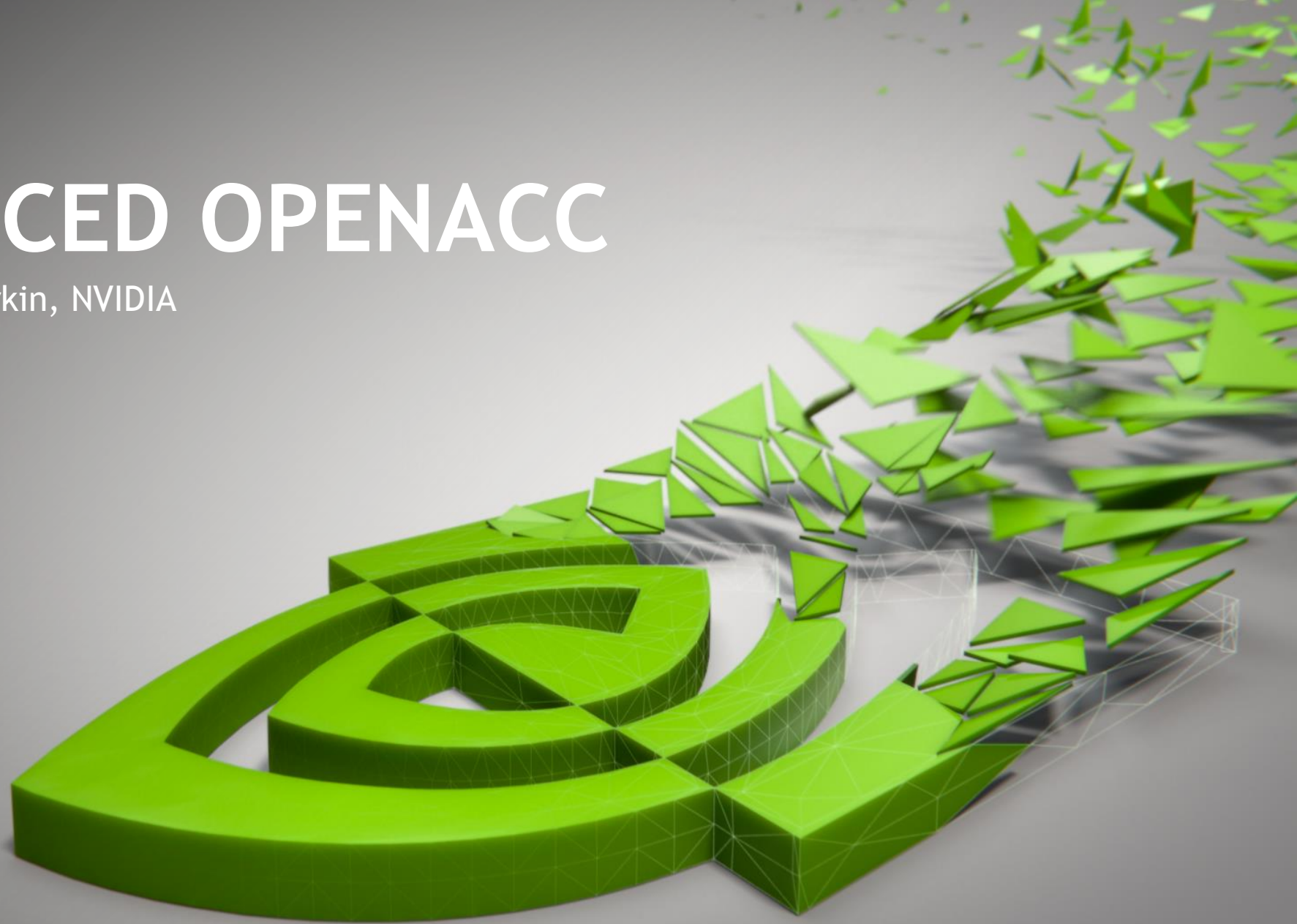
Nov 9: Advanced OpenACC

Recordings:

<https://developer.nvidia.com/intro-to-openacc-course-2016>

ADVANCED OPENACC

Lecture 3: Jeff Larkin, NVIDIA



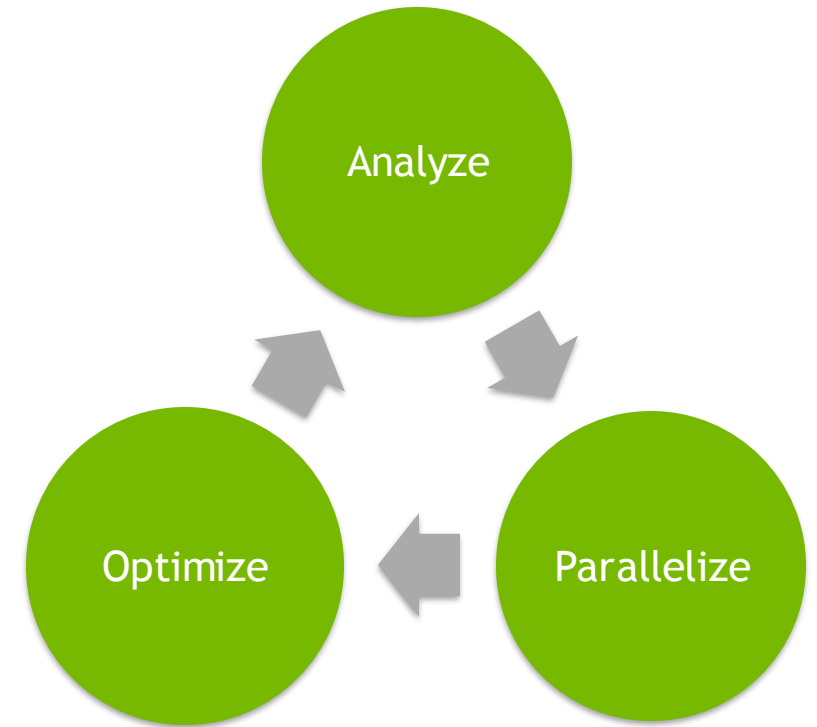
Today's Objectives

Understand several common refactoring techniques to improve performance

Understand the routine directive

Understand async and wait and how to apply them to data pipelining

Understand the OpenACC interoperability with libraries and CUDA



Additional Common Optimizations

The collapse Clause

collapse(*n*): Takes the next *n* tightly-nested loops, folds them into one, and applies the OpenACC directives to the new loop.

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



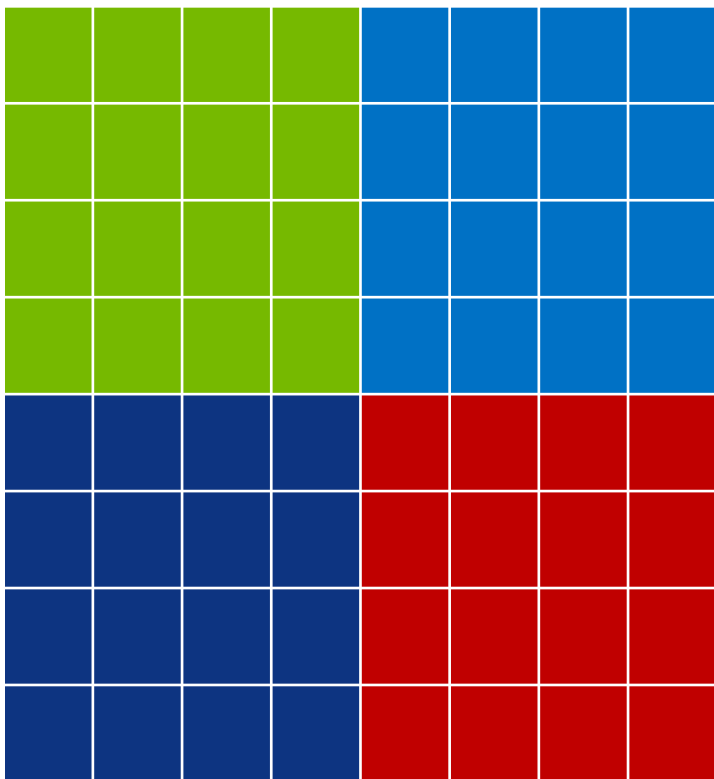
```
#pragma acc parallel loop  
for(int ij=0; ij<N*M; ij++)  
    ...
```

Why?

- Collapse outer loops to enable creating more gangs.
- Collapse inner loops to enable longer vector lengths.
- Collapse all loops, when possible, to do both.

The tile clause

Operate on smaller blocks of the operation to exploit data locality



```
#pragma acc loop tile(4,4)
for(i = 1; i <= ROWS; i++) {
    for(j = 1; j <= COLUMNS; j++) {
        Temp[i][j] = 0.25 *
            (Temp_last[i+1][j] +
             Temp_last[i-1][j] +
             Temp_last[i][j+1] +
             Temp_last[i][j-1]);
    }
}
```


Stride-1 Memory Accesses

```
for(i=0; i<N; i++)  
  for(j=0; j<M; j++)  
  {  
    A[i][j][1] = 1.0f;  
    A[i][j][2] = 0.0f;  
  }  
}
```

The fastest dimension is length 2
and fastest loop strides by 2.

```
for(i=0; i<N; i++)  
  for(j=0; j<M; j++)  
  {  
    A[1][i][j] = 1.0f;  
    A[2][i][j] = 0.0f;  
  }  
}
```

Now the inner loop is the fastest
dimension through memory.

Stride-1 Memory Accesses

```
for(i=0; i<N; i++)  
  for(j=0; j<M; j++)  
  {  
    A[i][j].a = 1.0f;  
    A[i][j].b = 0.0f;  
  }  
}
```

If all threads access the “a” element, they will be accessing every-other memory element.

```
for(i=0; i<N; i++)  
  for(j=0; j<M; j++)  
  {  
    Aa[i][j] = 1.0f;  
    Ab[i][j] = 0.0f;  
  }  
}
```

Now all threads are accessing contiguous elements of Aa and Ab.

OpenACC Routine Directive

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.

You must declare one level of parallelism on the routine directive.

Routine Directive: C/C++

```
// foo.h
#pragma acc routine seq
double foo(int i);

// Used in main()
#pragma acc parallel loop
for(int i=0;i<N;i++) {
    array[i] = foo(i);
}
```

- ▶ 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 Routine: Fortran

```
module foo_mod
  contains
  real(8) function foo(i)
    implicit none
    !$acc routine(foo) seq
    integer,intent(in),value :: i
    ...
  end function foo
end module foo_mod
```

The **routine** directive may appear in a Fortran function or subroutine definition, or in an interface block.

The save attribute is not supported.

Nested acc routines require the routine directive within each nested routine.

Asynchronous Programming with OpenACC

Asynchronous Programming

Programming multiple operations without immediate synchronization

Real World Examples:

- Cooking a Meal: Boiling potatoes while preparing other parts of the dish.
- Three students working on a project on George Washington, one researches his early life, another his military career, and the third his presidency.
- Automobile assembly line: each station adds a different part to the car until it is finally assembled.

Asynchronous OpenACC

So far, all OpenACC directives have been synchronous with the host

- Host waits for the parallel loop to complete
- Host waits for data updates to complete

Most OpenACC directives can be made asynchronous

- Host issues multiple parallel loops to the device before waiting
- Host performs part of the calculation while the device is busy
- Data transfers can happen before the data is needed

Asynchronous Pipelining

- ▶ Very large operations may frequently be broken into smaller parts that may be performed independently.
- ▶ **Pipeline Stage** -A single step, which is frequently limited to 1 part at a time



Photo by Roger Wollstadt, used via Creative Commons

Case Study: Image Filter

The example code reads an image from a file, applies a simple filter to the image, then outputs the file.

Skills Used:

- Parallelize the filter on the GPU
- Data Region and Update Directive
- Async and Wait directives
- Pipelining



Image Filter Code

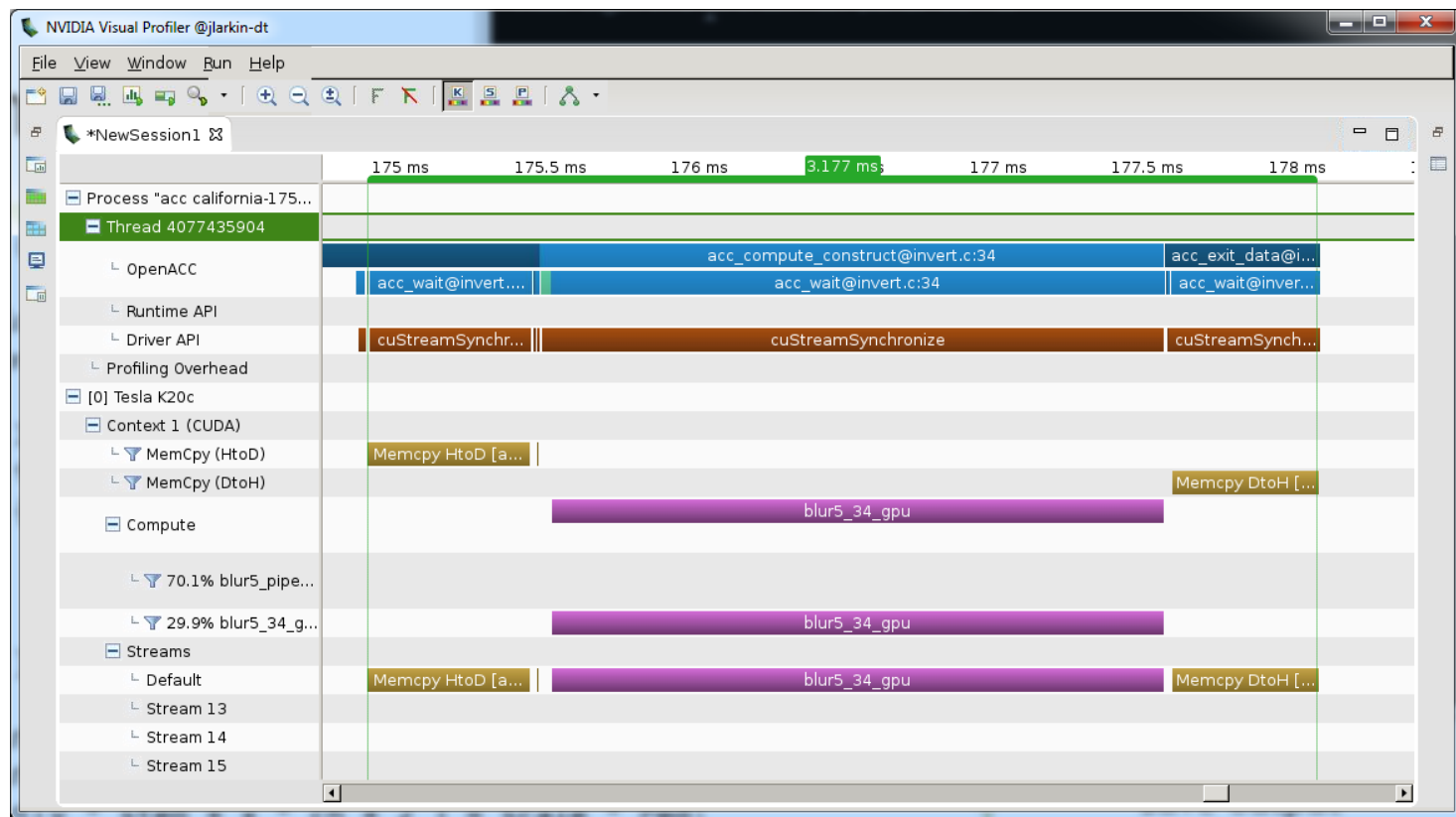
```
#pragma acc parallel loop collapse(2) gang vector
for ( y = 0; y < h; y++ ); for ( x = 0; x < w; x++ )
{
    float blue = 0.0, green = 0.0, red = 0.0;
    for ( int fy = 0; fy < filtersize; fy++ )
    {
        long iy = y - (filtersize/2) + fy;
        for ( int fx = 0; fx < filtersize; fx++ )
        {
            blue+=filter[fy][fx]*imgData[iy*step+ix*ch];
            green+=filter[fy][fx]*imgData[iy*step+ix*ch+1];
            red+=filter[fy][fx]*imgData[iy*step+ix*ch+2];
        }
    }
    out[y * step + x * ch]          = 255 - scale * blue;
    out[y * step + x * ch + 1 ] = 255 - scale * green;
    out[y * step + x * ch + 2 ] = 255 - scale * red;
}
```

Iterate over all pixels

Apply filter

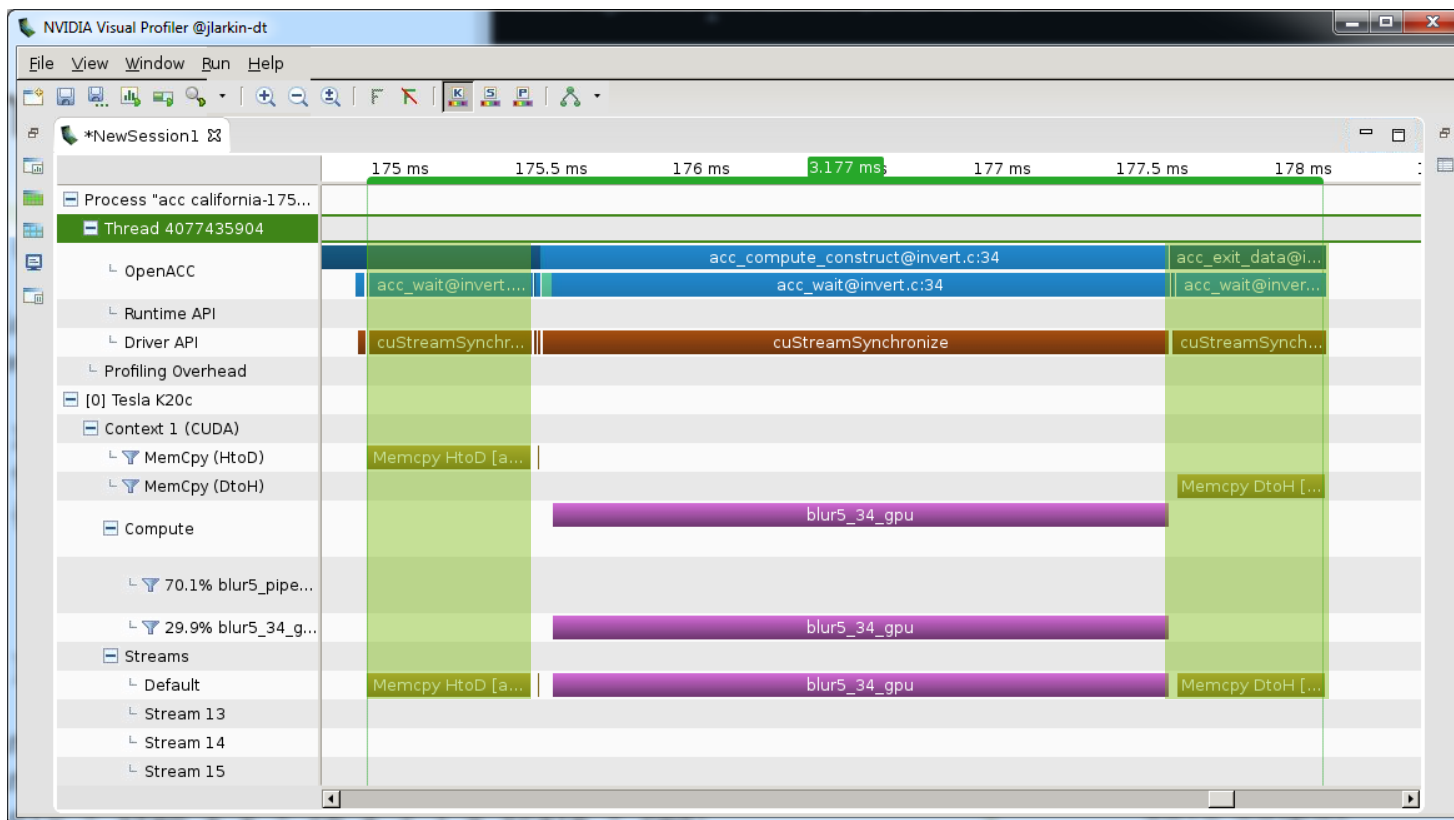
Save output

GPU Timeline



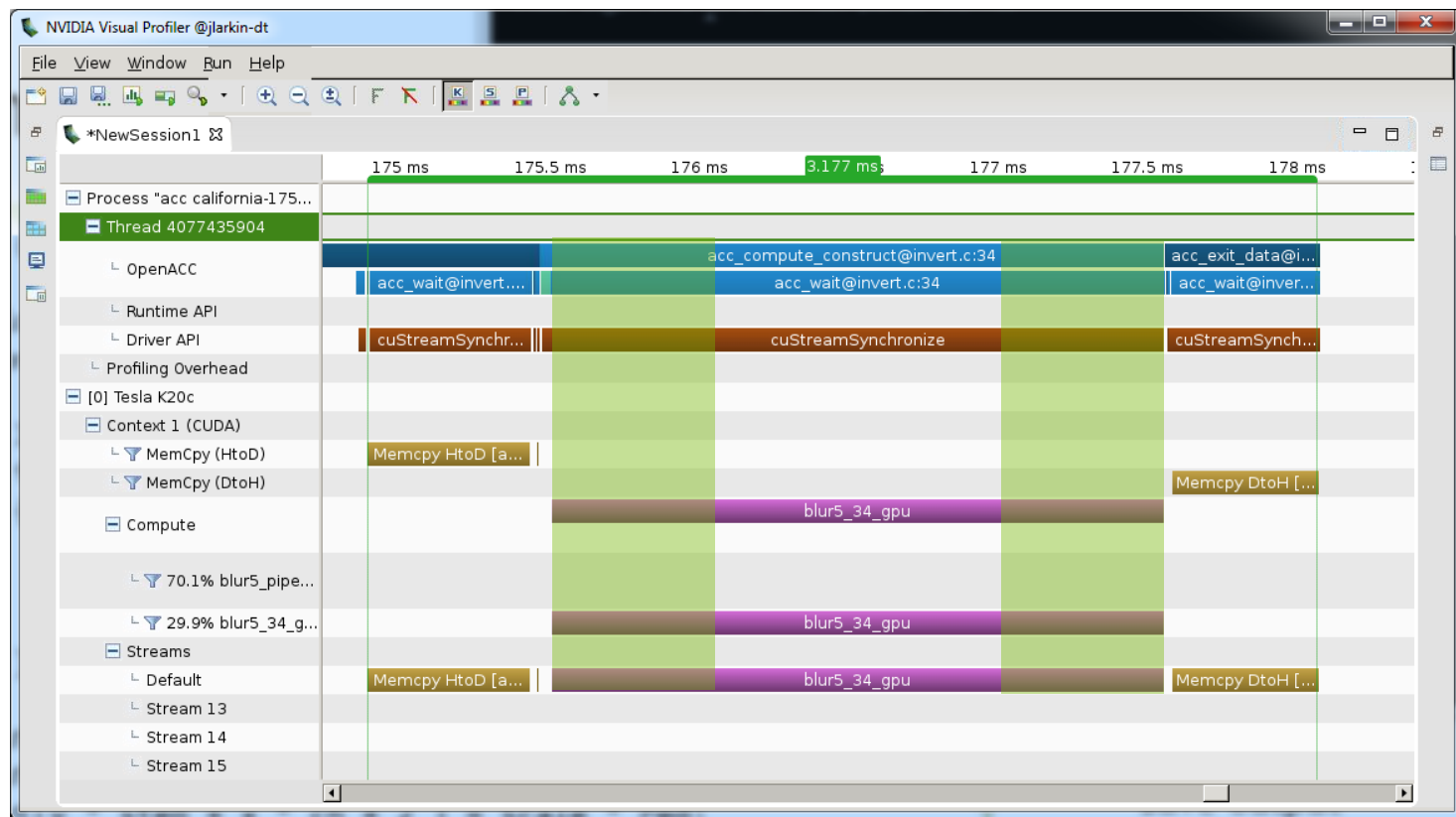
GPU Timeline

Roughly 1/3 of the runtime is occupied with data transfers.



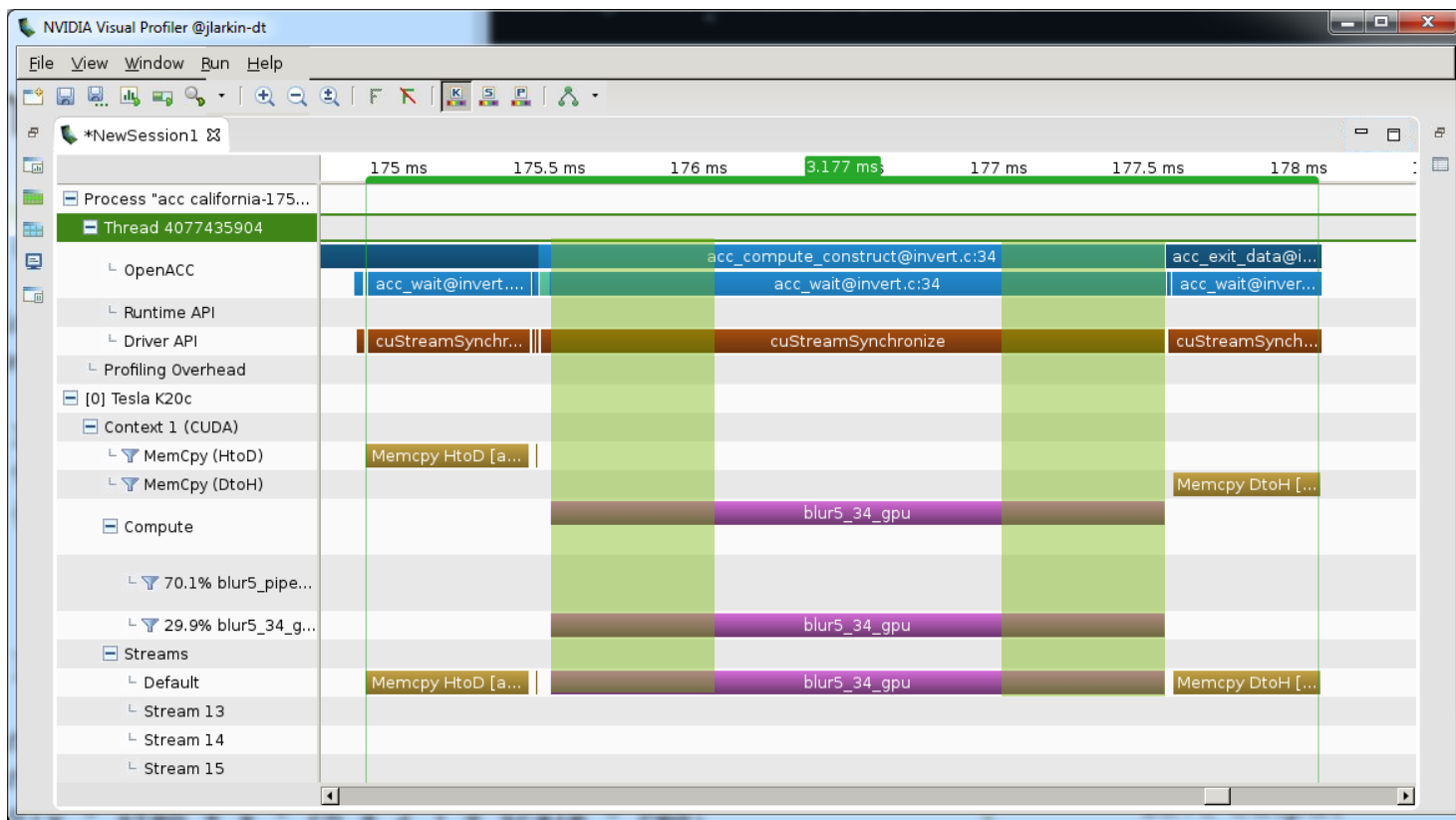
GPU Timeline

Roughly 1/3 of the runtime is occupied with data transfers.



What if we could overlap the copies with the computation?

GPU Timeline



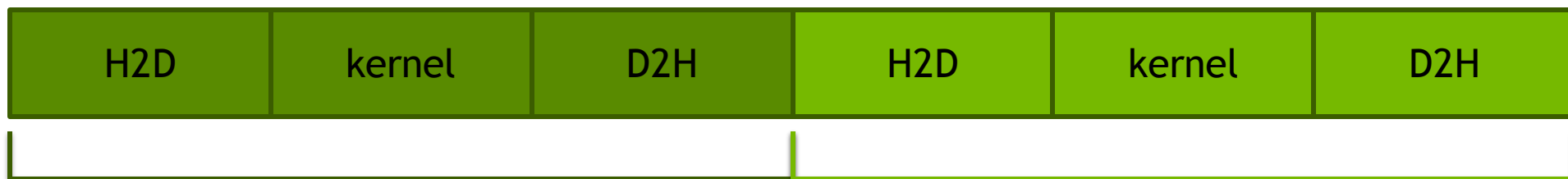
Roughly 1/3 of the runtime is occupied with data transfers.

What if we could overlap the copies with the computation?

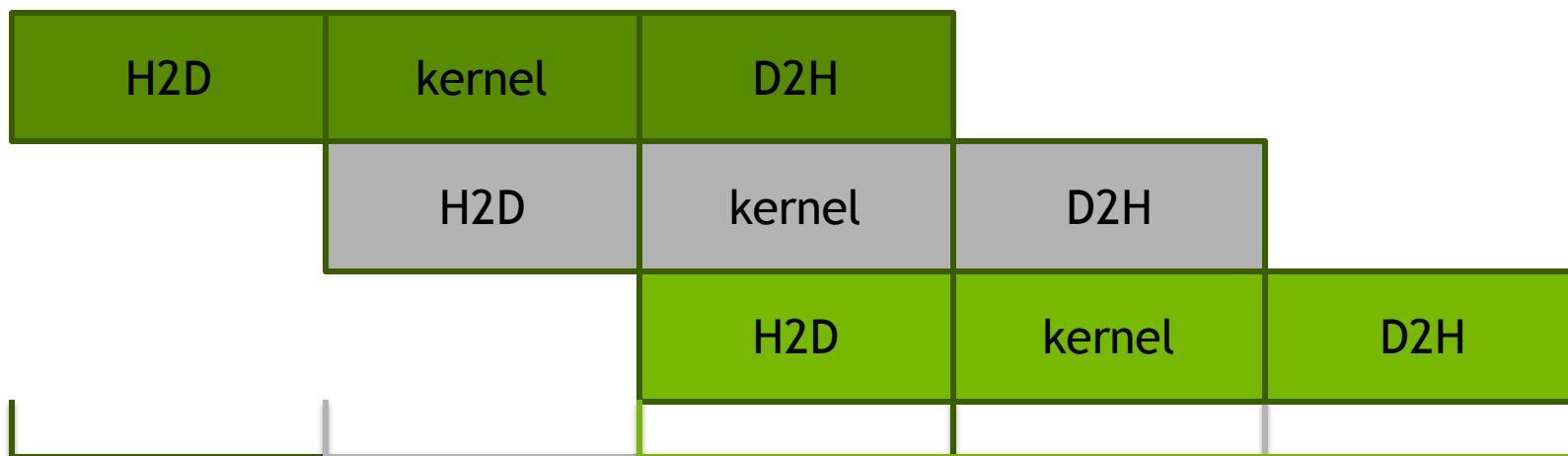
Rough Math:

$$\begin{aligned} 3.2\text{ms} - .5\text{ms} - .5\text{ms} \\ = 2.2\text{ ms} \\ 3.2 / 2.2 \approx 1.45X \end{aligned}$$

Pipelining Data Transfers



Two Independent Operations Serialized



Overlapping Copying and Computation

NOTE: In real applications, your boxes will not be so evenly sized.

Blocking the Code

Before we can overlap data transfers with computation, we need to break our work into smaller chunks.

Since each pixel is calculated independently, the work can be easily divided.

We'll divide along chunks of rows for convenience.



Blocked Image Filter Code

```
#pragma acc data copyin(imgData[:w*h*ch],filter)
                    copyout(out[:w*h*ch])
for ( long blocky = 0; blocky < nblocks; blocky++){
    long starty = blocky * blocksize;
    long endy   = starty + blocksize;
    #pragma acc parallel loop collapse(2) gang vector
    for (y=starty; y<endy; y++); for(x=0; x<w; x++ ) {
        float blue = 0.0, green = 0.0, red = 0.0;
        for ( int fy = 0; fy < filtersize; fy++ ) {
            long iy = y - (filtersize/2) + fy;
            for ( int fx = 0; fx < filtersize; fx++ ){
                long ix = x - (filtersize/2) + fx;
                blue+=filter[fy][fx]*imgData[iy*step+ix*ch];
                green+=filter[fy][fx]*imgData[iy*step+ix*ch+1];
                red+=filter[fy][fx]*imgData[iy*step+ix*ch+2];
            }
        }
        out[y*step+x*ch]    = 255 - (scale * blue);
        out[y*step+x*ch+1] = 255 - (scale * green);
        out[y*step+x*ch+2] = 255 - (scale * red);
    }
}
```

1. Add blocking loop

Blocked Image Filter Code

```
#pragma acc data copyin(imgData[:w*h*ch],filter)
                    copyout(out[:w*h*ch])
for ( long blocky = 0; blocky < nblocks; blocky++){
    long starty = blocky * blocksize;
    long endy   = starty + blocksize;
    #pragma acc parallel loop collapse(2) gang vector
    for (y=starty; y<endy; y++); for(x=0; x<w; x++) {
        float blue = 0.0, green = 0.0, red = 0.0;
        for ( int fy = 0; fy < filtersize; fy++ ) {
            long iy = y - (filtersize/2) + fy;
            for ( int fx = 0; fx < filtersize; fx++ ){
                long ix = x - (filtersize/2) + fx;
                blue+=filter[fy][fx]*imgData[iy*step+ix*ch];
                green+=filter[fy][fx]*imgData[iy*step+ix*ch+1];
                red+=filter[fy][fx]*imgData[iy*step+ix*ch+2];
            }
        }
        out[y*step+x*ch]   = 255 - (scale * blue);
        out[y*step+x*ch+1] = 255 - (scale * green);
        out[y*step+x*ch+2] = 255 - (scale * red);
    }
}
```

1. Add blocking loop

2. Adjust “y” to only
iterate through rows within
a single chunk.

Blocked Image Filter Code

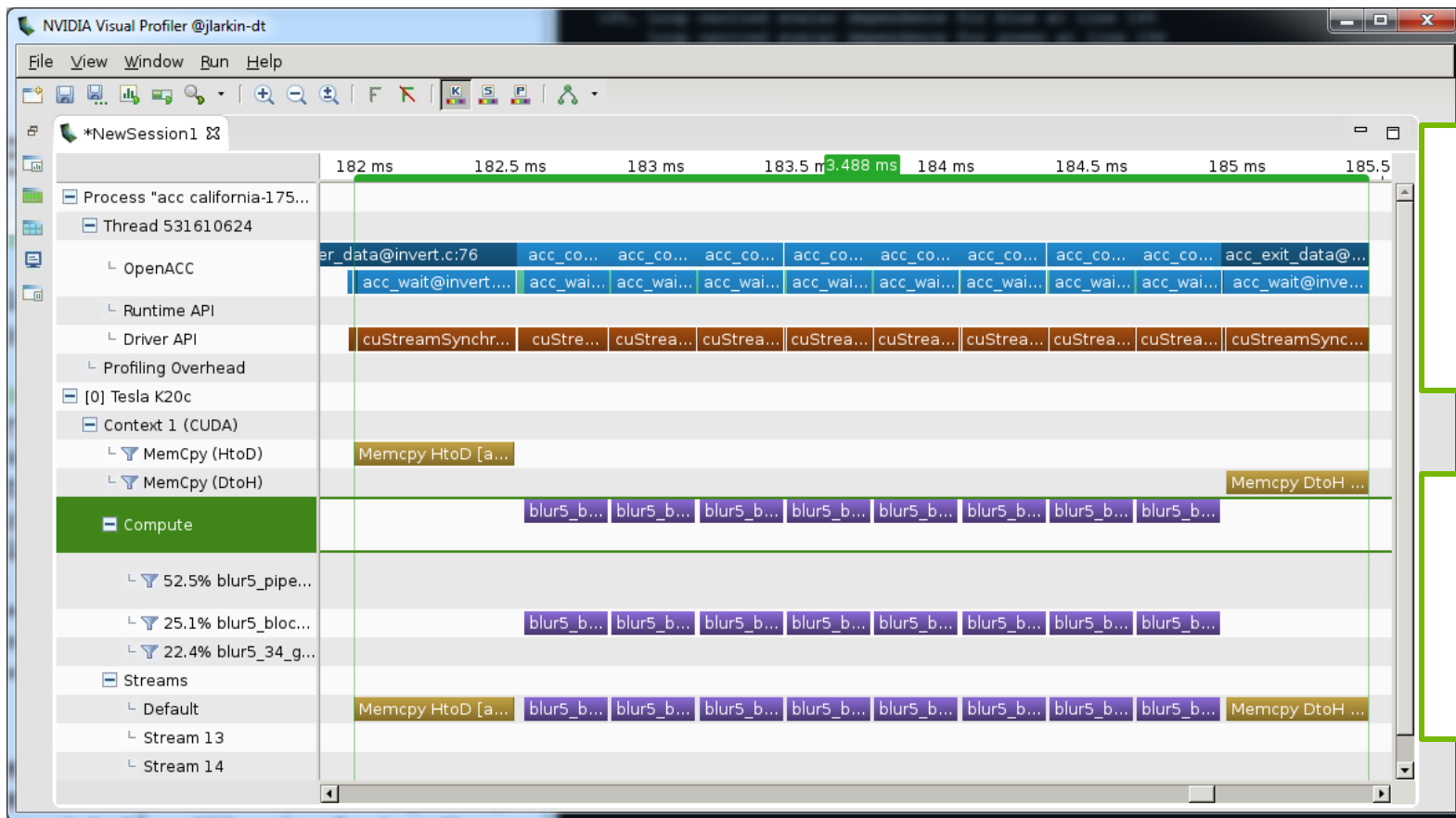
```
#pragma acc data copyin(imgData[:w*h*ch],filter)
                    copyout(out[:w*h*ch])
for ( long blocky = 0; blocky < nblocks; blocky++){
    long starty = blocky * blocksize;
    long endy   = starty + blocksize;
    #pragma acc parallel loop collapse(2) gang vector
    for (y=starty; y<endy; y++); for(x=0; x<w; x++) {
        float blue = 0.0, green = 0.0, red = 0.0;
        for ( int fy = 0; fy < filtersize; fy++ ) {
            long iy = y - (filtersize/2) + fy;
            for ( int fx = 0; fx < filtersize; fx++ ){
                long ix = x - (filtersize/2) + fx;
                blue+=filter[fy][fx]*imgData[iy*step+ix*ch];
                green+=filter[fy][fx]*imgData[iy*step+ix*ch+1];
                red+=filter[fy][fx]*imgData[iy*step+ix*ch+2];
            }
        }
        out[y*step+x*ch]    = 255 - (scale * blue);
        out[y*step+x*ch+1] = 255 - (scale * green);
        out[y*step+x*ch+2] = 255 - (scale * red);
    }
}
```

3. Data region to handle
copies

1. Add blocking loop

2. Adjust “y” to only
iterate through rows within
a single chunk.

GPU Timeline Blocked



Compute kernel is now broken in 8 blocks.

Data transfers still happen at beginning and end.

OpenACC Update Directive

Programmer specifies an array (or part of an array) that should be refreshed within a data region. (Host and Device copies are made coherent)

```
do_something_on_device()
```

```
!$acc update host(a)
```



Copy “a” from GPU to CPU

```
do_something_on_host()
```

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



Copy “a” from CPU to GPU

Note: Update “host” has been deprecated and renamed “self”

Blocked Update Code

Change data clauses to
create

```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
                    copyin(filter)
for ( long blocky = 0; blocky < nblocks; blocky++)
{
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy   = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update
device(imgData[starty*step:blocksize*step])
    starty = blocky * blocksize;
    endy = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector
    for (y=starty; y<endy; y++) for ( x=0; x<w; x++ ) {
        <filter code omitted>
        out[y * step + x * ch]      = 255 - (scale * blue);
        out[y * step + x * ch + 1 ] = 255 - (scale * green);
        out[y * step + x * ch + 2 ] = 255 - (scale * red);
    }
#pragma acc update self(out[starty*step:blocksize*step])
}
```

Blocked Update Code

Change data clauses to
create

```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
                    copyin(filter)
for ( long blocky = 0; blocky < nblocks; blocky++)
{
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy   = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update
device(imgData[starty*step:blocksize*step])
    starty = blocky * blocksize;
    endy = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector
    for (y=starty; y<endy; y++) for ( x=0; x<w; x++ ) {
        <filter code omitted>
        out[y * step + x * ch]      = 255 - (scale * blue);
        out[y * step + x * ch + 1 ] = 255 - (scale * green);
        out[y * step + x * ch + 2 ] = 255 - (scale * red);
    }
#pragma acc update self(out[starty*step:blocksize*step])
}
```

Update data one block at a
time.

Blocked Update Code

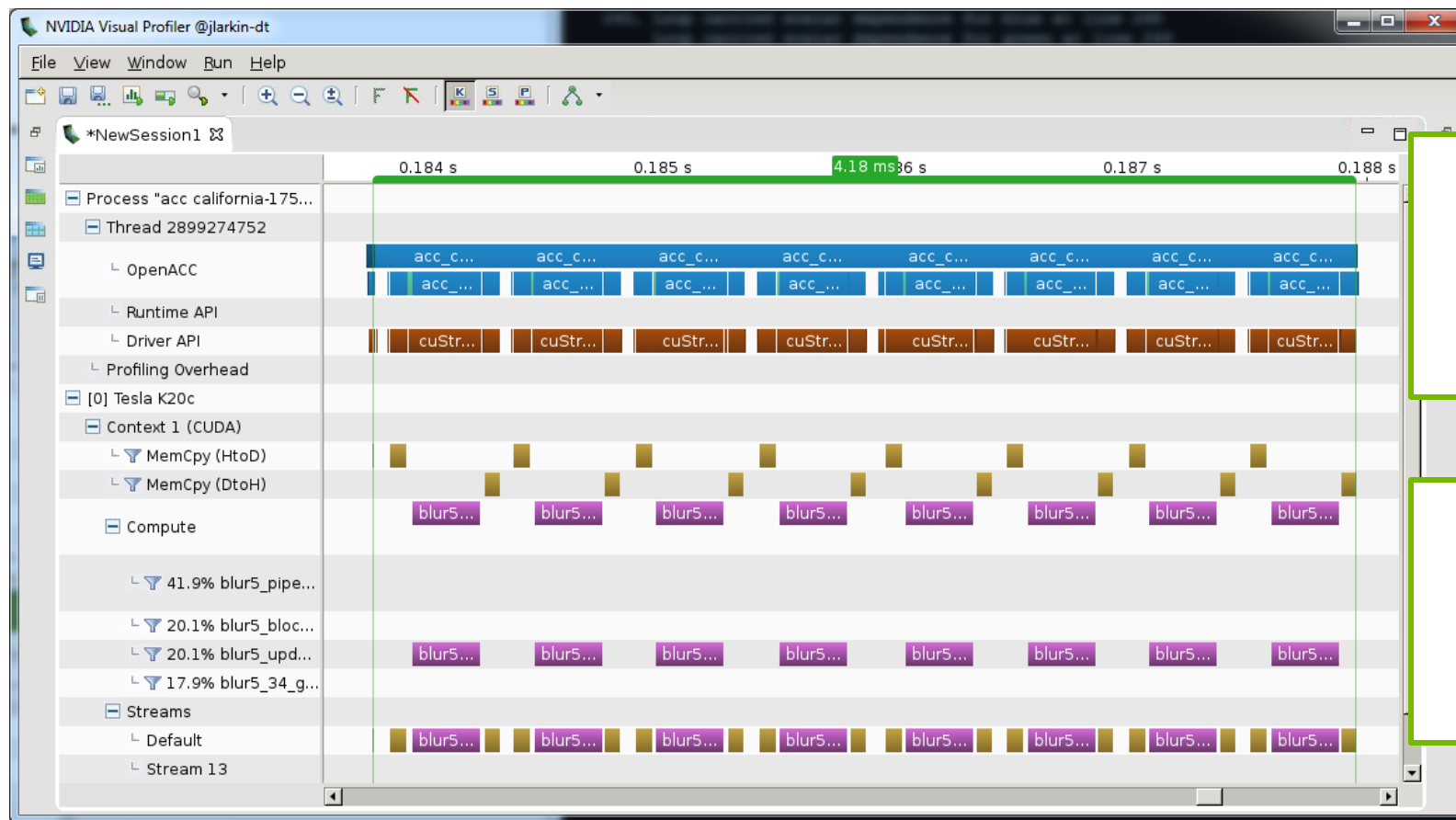
Change data clauses to
create

```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
                    copyin(filter)
for ( long blocky = 0; blocky < nblocks; blocky++)
{
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy   = MIN(h,starty + blocksize + filtersize/2);
    #pragma acc update
    device(imgData[starty*step:blocksize*step])
        starty = blocky * blocksize;
        endy = starty + blocksize;
    #pragma acc parallel loop collapse(2) gang vector
        for (y=starty; y<endy; y++) for ( x=0; x<w; x++ ) {
            <filter code omitted>
            out[y * step + x * ch]      = 255 - (scale * blue);
            out[y * step + x * ch + 1 ] = 255 - (scale * green);
            out[y * step + x * ch + 2 ] = 255 - (scale * red);
        }
    #pragma acc update self(out[starty*step:blocksize*step])
}
```

Update data one block at a
time.

Copy results back one block
at a time.

GPU Timeline Blocked Updates



Compute and Updates happen in blocks.

The last step is to overlap compute and copy.

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 and enables pipelining and concurrent operations

```
#pragma acc parallel loop async(1)
...
#pragma acc parallel loop async(1)
for(int i=0; i<N; i++)
    ...
#pragma acc wait(1)
for(int i=0; i<N; i++)
```

If *n* is not specified, *async* will go into a default queue and *wait* will wait all previously queued work.

Pipelined Code

```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
    copyin(filter)
{
for ( long blocky = 0; blocky < nblocks; blocky++)
{
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy    = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update device(imgData[starty*step:blocksize*step]) async(block%3+1)
    starty = blocky * blocksize;
    endy = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector async(block%3+1)
    for (y=starty; y<endy; y++) for ( x=0; x<w; x++ ) {
        <filter code omitted>
        out[y * step + x * ch]          = 255 - (scale * blue);
        out[y * step + x * ch + 1 ] = 255 - (scale * green);
        out[y * step + x * ch + 2 ] = 255 - (scale * red);
    }
#pragma acc update self(out[starty*step:blocksize*step]) async(block%3+1)
}
#pragma acc wait
}
```

Cycle between 3 async
queues by blocks.

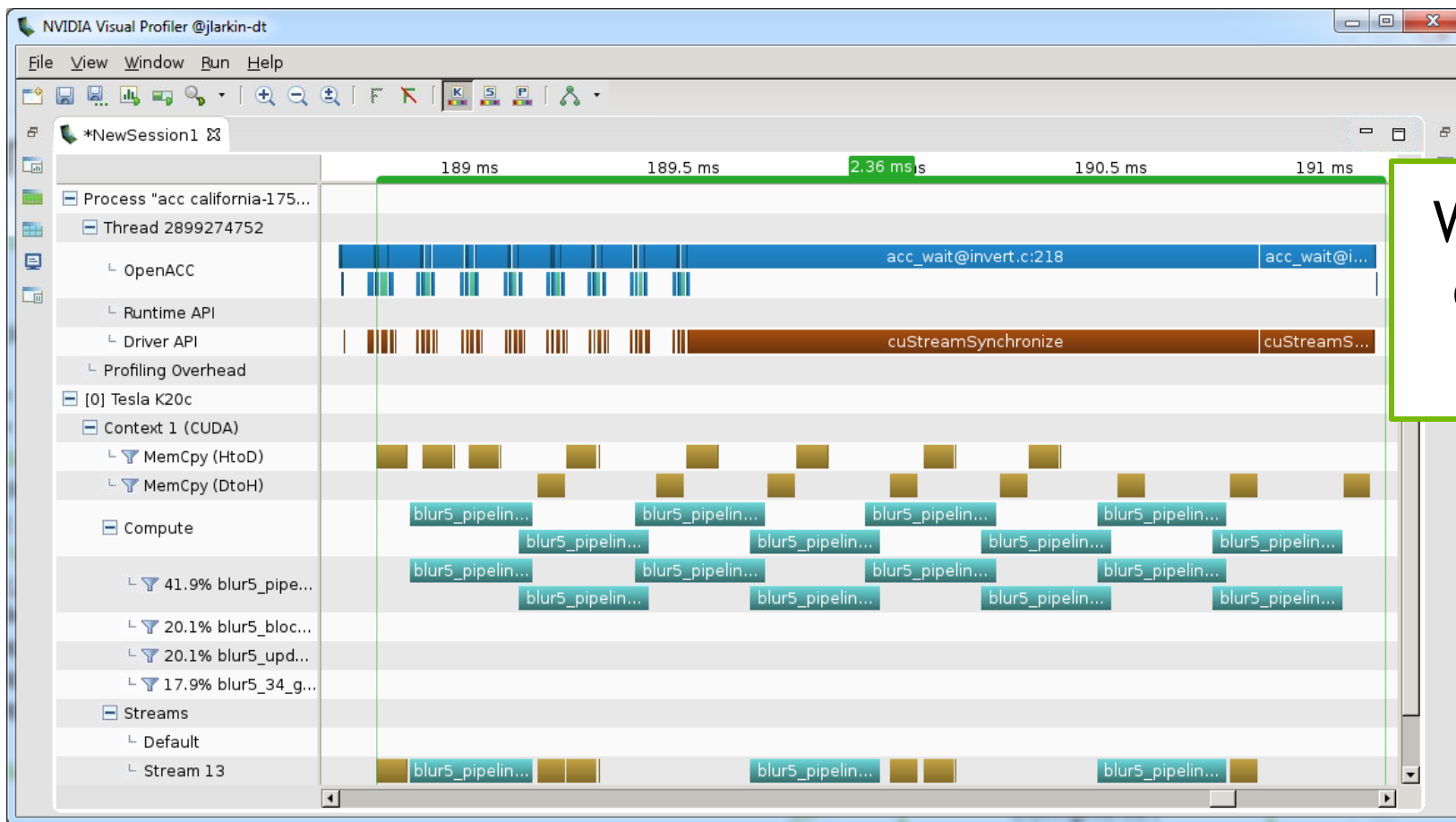
Pipelined Code

```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
    copyin(filter)
{
for ( long blocky = 0; blocky < nblocks; blocky++)
{
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy    = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update device(imgData[starty*step:blocksize*step]) async(block%3+1)
    starty = blocky * blocksize;
    endy = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector async(block%3+1)
    for (y=starty; y<endy; y++) for ( x=0; x<w; x++ ) {
        <filter code omitted>
        out[y * step + x * ch]      = 255 - (scale * blue);
        out[y * step + x * ch + 1 ] = 255 - (scale * green);
        out[y * step + x * ch + 2 ] = 255 - (scale * red);
    }
#pragma acc update self(out[starty*step:blocksize*step]) async(block%3+1)
}
#pragma acc wait
}
```

Cycle between 3 async queues by blocks.

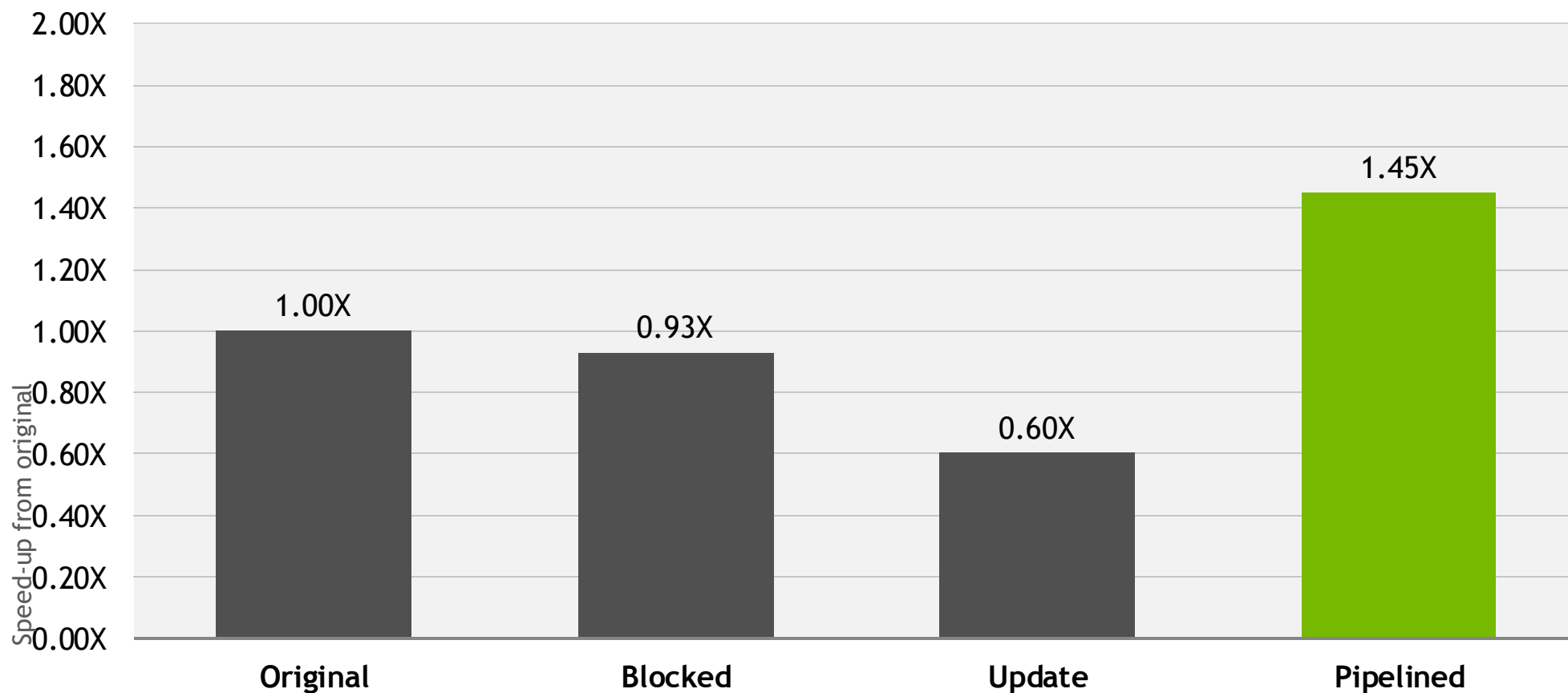
Wait for all blocks to complete.

GPU Timeline Pipelined



We're now able to overlap compute and copy.

Step-by-Step Performance



Multi-GPU Pipelining

Multi-GPU OpenACC with OpenMP

```
#pragma omp parallel
{
    int my_gpu = omp_get_thread_num();
    acc_set_device_num(my_gpu, acc_device_nvidia);

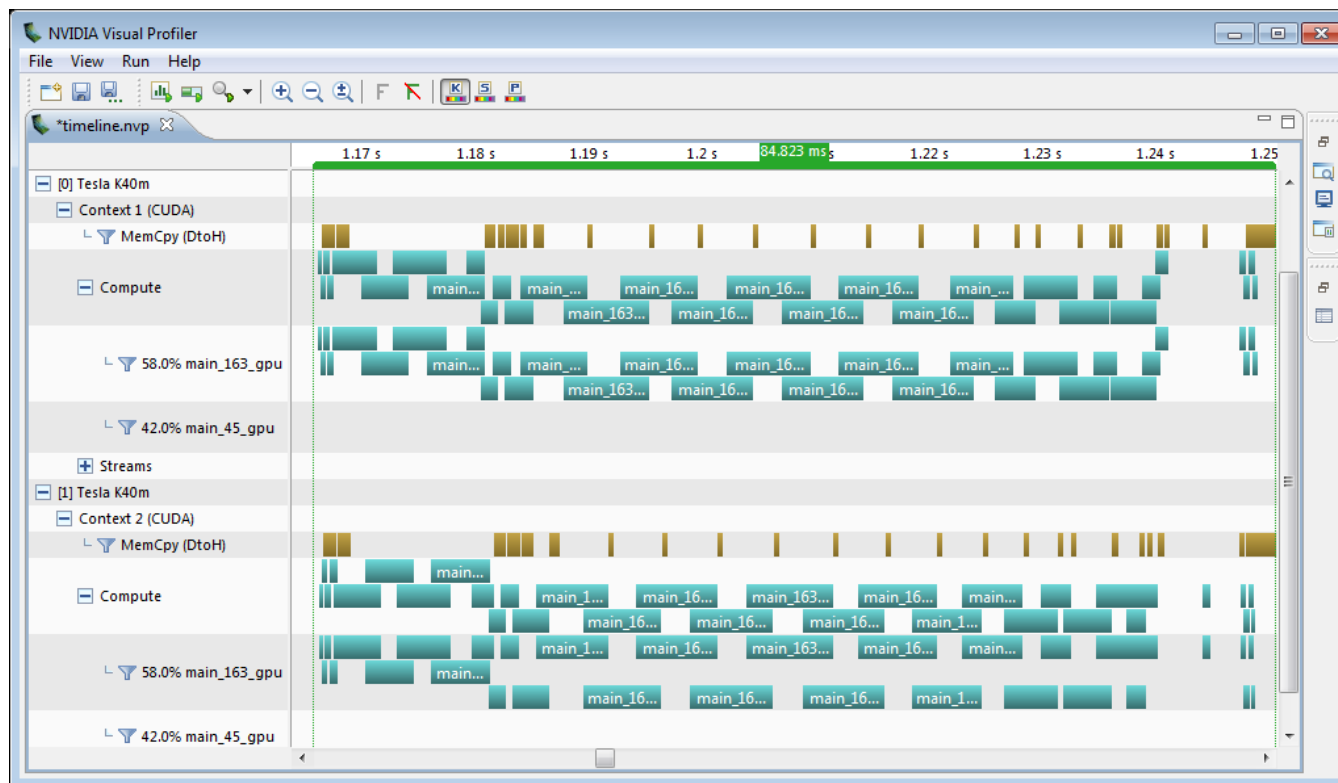
    #pragma acc data create(image[0:HEIGHT*WIDTH])
    {
        int queue = 1;
        #pragma omp for schedule(static,1) firstprivate(queue)
        for(int block=0; block < numblocks; block++)
        {
            int ystart = block * blocksize;
            int yend    = ystart + blocksize;
            #pragma acc parallel loop async(queue)
            for(int y=ystart; y<yend; y++) {
                for(int x=0; x<WIDTH; x++) {
                    image[y*WIDTH+x]=mandelbrot(x,y);
                }
            }
            #pragma acc update self(image[ystart*WIDTH:WIDTH*blocksize]) async(queue)
            queue = queue%2+1;
        }
        #pragma acc wait
    }
}
```

Set the device number, all work will be sent to this device.

Use multiple queues per device to get copy compute overlap

Wait for all work to complete (per device)

Multi-GPU Mandelbrot Profile



OpenACC Interoperability

OpenACC Interoperability

OpenACC plays well with others.

Add CUDA or accelerated libraries to an OpenACC application

Add OpenACC to an existing accelerated application

Share data between OpenACC and CUDA

The screenshot shows the NVIDIA Developer Zone website. At the top is the NVIDIA logo and 'DEVELOPER ZONE' text. Below this is a navigation bar with links for DEVELOPER CENTERS, TECHNOLOGIES, TOOLS, RESOURCES, and COMMUNITY. A search bar is on the right. The main header is 'CUDA ZONE'. Below this is a large banner for the 'GPU TECHNOLOGY CONFERENCE' with the text 'Explore the world's biggest GPU developer conference.' and a 'LEARN MORE' button. The main content area is titled 'EXPLORE CUDA ZONE' and contains several sections: 'WHAT IS CUDA' (Learn more about the CUDA parallel computing platform and programming model.), 'GET STARTED - PARALLEL COMPUTING' (Find out about different paths and options for deploying CUDA and GPU computing in your project.), 'CUDA IN ACTION - RESEARCH & APPS' (Supercomputing is now accessible for every researcher and scientist. Find latest research, applications and links to how CUDA is transforming the industry.), 'CUDA TOOLKIT' (The NVIDIA CUDA Toolkit provides a comprehensive development environment for C and C++ developers building GPU-accelerated applications.), 'CUDA EDUCATION & TRAINING' (Get educated with online courses, webinars, University Courses and wealth of technical papers & documentation.), and 'CUDA TOOLS & ECOSYSTEM' (Learn more about powerful CUDA tools, libraries, languages, and other development aids available from NVIDIA & partners.). On the right side, there are three vertical sections: 'QUICKLINKS' (CUDA Downloads, CUDA GPUs, NVIDIA Nsight Visual Studio Edition, Get Started - Parallel Computing, CUDA Tools & Ecosystem, CUDA FAQ), 'NVIDIA DEVELOPER PROGRAMS' (Get exclusive access to the latest software, report bugs and receive notifications for special events., LEARN MORE AND REGISTER), and 'LATEST NEWS' (NVIDIA Nsight Visual Studio Edition 3.2 Available Now With Windows 8.1 Support And Improved DirectCompute Profiling, OpenACC Training: Nov 5th, Nsight Visual Studio Edition 3.1 Final Now Available With Visual Studio 2012, DirectX 11.1 And CUDA 5.5 Support!, Robotics Expert Starts A New Facebook GPU Computing Community, CUDA 5.5 Production Release - Now Available, More). At the bottom, there is a 'PARALLEL FOR ALL BLOG' section with a post titled '5 Things You Should Know About the New Maxwell GPU Architecture' dated February 21, 2014, and a link to 'CUDACasts Episode 17: Unstructured Data Lifetimes in OpenACC 2.0'.

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

host_data Example

OpenACC Main

```
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
```

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

CUDA C Kernel & Wrapper

```
__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);
}
```

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

CUBLAS Library & OpenACC

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

This includes...

- CUBLAS
- Libsci_acc
- CUFFT
- MAGMA
- CULA
- Thrust
- ...

OpenACC Main Calling CUBLAS

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

cublasInit();

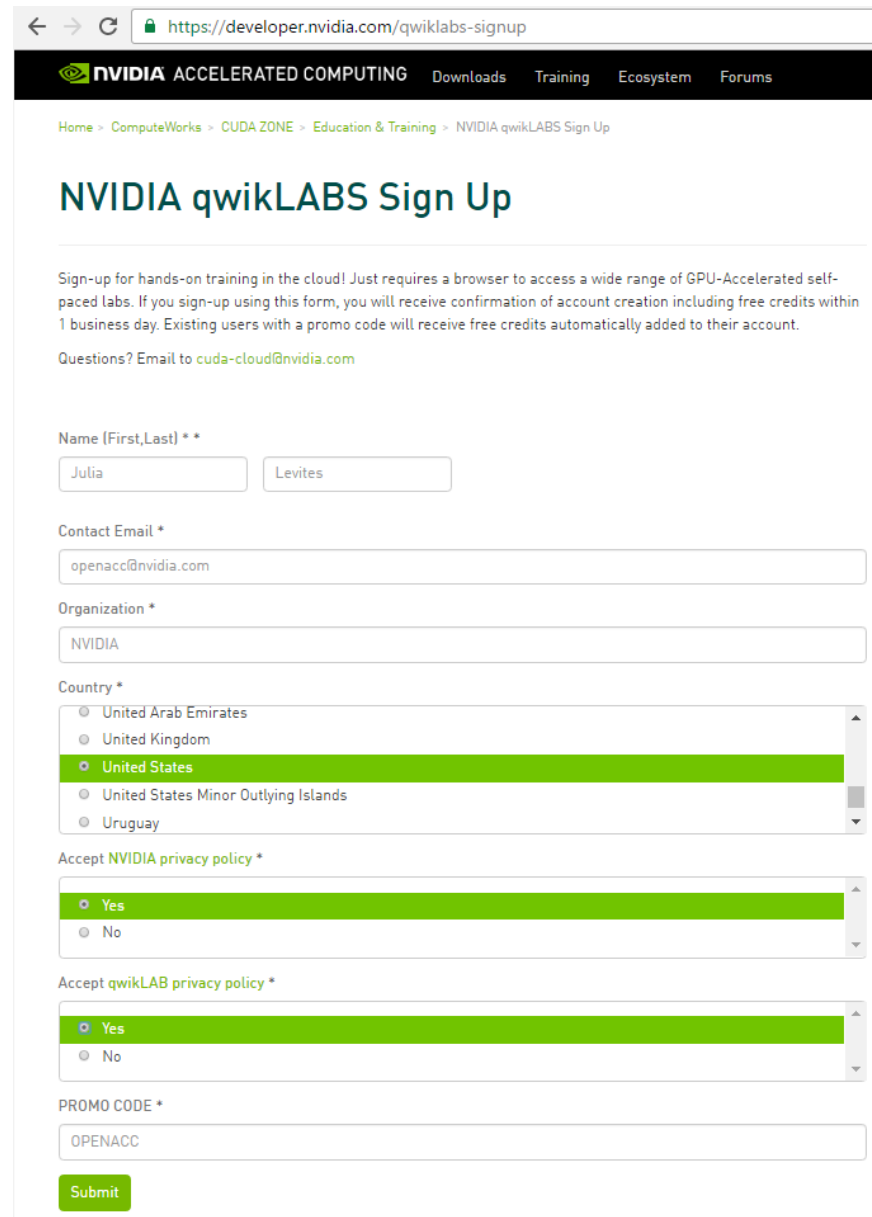
#pragma acc data copyin(x[0:N]) copy(y[0:N])
{
    #pragma acc host_data use_device(x,y)
    {
        cublasSaxpy(N, 2.0, x, 1, y, 1);
    }
}

cublasShutdown();
```

Using QwikLabs

Getting access

1. Create an account with NVIDIA qwikLABS <https://developer.nvidia.com/qwiklabs-signup>
2. Enter a promo code OPENACC16 before submitting the form
3. Free credits will be added to your account
4. Start using OpenACC!



The screenshot shows the NVIDIA qwikLABS Sign Up page. The browser address bar displays <https://developer.nvidia.com/qwiklabs-signup>. The page header includes the NVIDIA logo and navigation links: Accelerated Computing, Downloads, Training, Ecosystem, and Forums. A breadcrumb trail reads: Home > ComputeWorks > CUDA ZONE > Education & Training > NVIDIA qwikLABS Sign Up.

NVIDIA qwikLABS Sign Up

Sign-up for hands-on training in the cloud! Just requires a browser to access a wide range of GPU-Accelerated self-paced labs. If you sign-up using this form, you will receive confirmation of account creation including free credits within 1 business day. Existing users with a promo code will receive free credits automatically added to their account.

Questions? Email to cuda-cloud@nvidia.com

Name (First,Last) * *

Julia Levites

Contact Email *

openacc@nvidia.com

Organization *

NVIDIA

Country *

- United Arab Emirates
- United Kingdom
- United States**
- United States Minor Outlying Islands
- Uruguay

Accept NVIDIA privacy policy *

- Yes**
- No

Accept qwikLAB privacy policy *

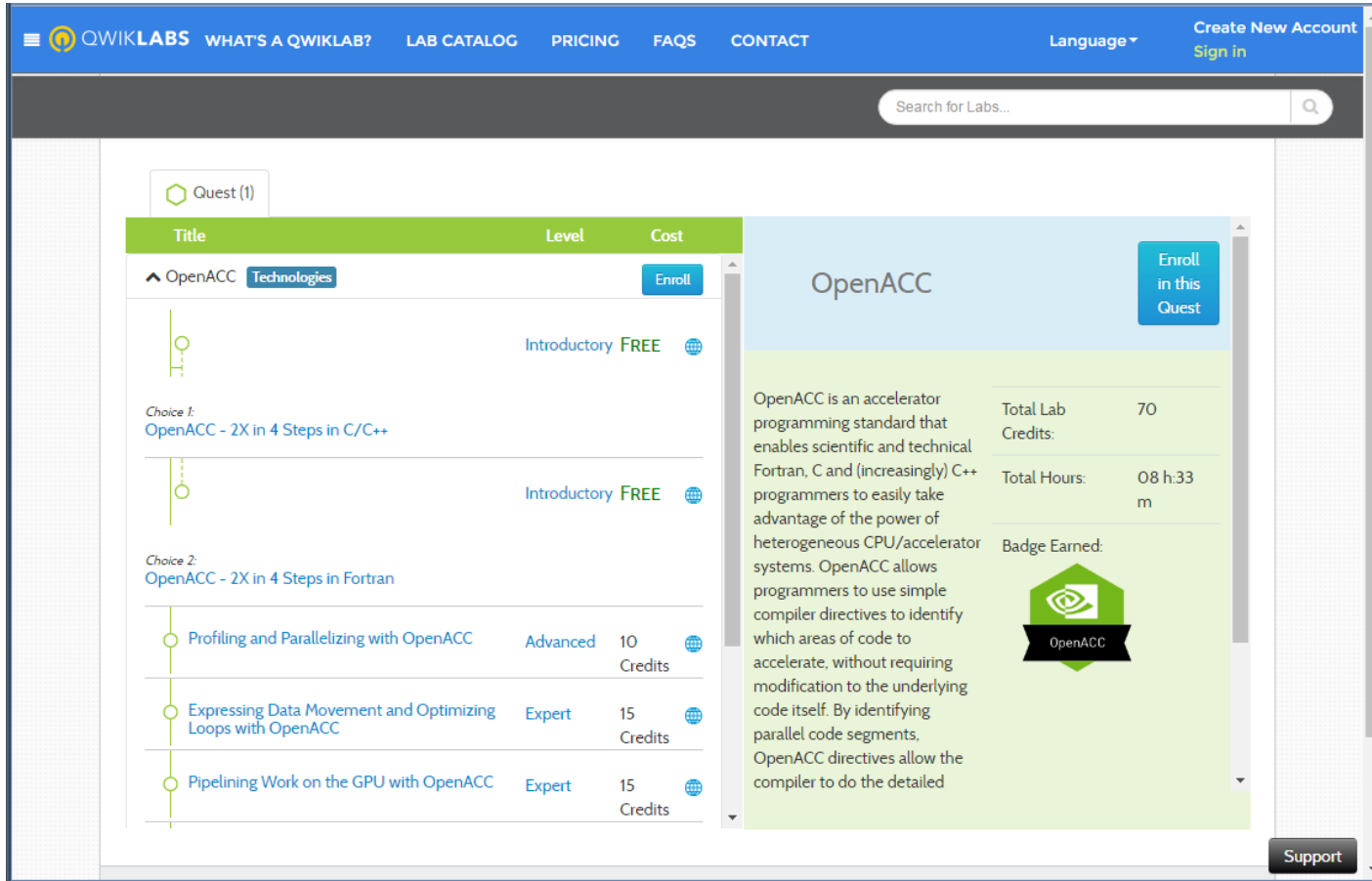
- Yes**
- No

PROMO CODE *

OPENACC

Submit

This week's labs



The screenshot shows the QwikLabs website interface. At the top, there is a navigation bar with links: QWIKLABS, WHAT'S A QWIKLAB?, LAB CATALOG, PRICING, FAQs, CONTACT, Language, and Create New Account / Sign in. Below the navigation bar is a search bar labeled "Search for Labs...".

The main content area is divided into two columns. The left column displays a list of labs under the heading "Quest (1)". The labs are:

- OpenACC Technologies** (Introductory, FREE) - Choice 1: OpenACC - 2X in 4 Steps in C/C++
- OpenACC - 2X in 4 Steps in Fortran** (Introductory, FREE)
- Profiling and Parallelizing with OpenACC** (Advanced, 10 Credits)
- Expressing Data Movement and Optimizing Loops with OpenACC** (Expert, 15 Credits)
- Pipelining Work on the GPU with OpenACC** (Expert, 15 Credits)

The right column displays the details for the **OpenACC** lab. It includes a description: "OpenACC is an accelerator programming standard that enables scientific and technical Fortran, C and (increasingly) C++ programmers to easily take advantage of the power of heterogeneous CPU/accelerator systems. OpenACC allows programmers to use simple compiler directives to identify which areas of code to accelerate, without requiring modification to the underlying code itself. By identifying parallel code segments, OpenACC directives allow the compiler to do the detailed". It also shows a table with lab statistics:

Metric	Value
Total Lab Credits:	70
Total Hours:	08 h:33 m

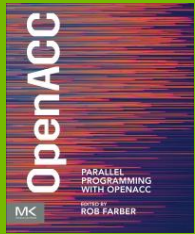
Below the table, it says "Badge Earned:" and shows the OpenACC badge. An "Enroll in this Quest" button is visible in the top right corner of the lab details section. A "Support" button is located at the bottom right of the page.

This week you should complete the “Pipelining Work on the GPU with OpenACC” lab. Please send questions to openacc@nvidia.com.

CERTIFICATION

Available after November 9th

1. Attend live lectures
2. Complete the test
3. Enter for a chance to win a Titan X or an OpenACC Book



Official rules:

<http://developer.download.nvidia.com/compute/OpenACC-Toolkit/docs/TITANX-GIVEAWAY-OPENACC-Official-Rules-2016.pdf>

OPENACC TOOLKIT

Free for Academia

Download link:

<https://developer.nvidia.com/openacc-toolkit>

NEW OPENACC BOOK

Parallel Programming with OpenACC

Now Available:

<http://store.elsevier.com/Parallel-Programming-with-OpenACC/Rob-Farber/isbn-9780124103979/>

Where to find help

- OpenACC Course Recordings - <https://developer.nvidia.com/openacc-courses>
- PGI Website - <http://www.pgroup.com/resources>
- OpenACC on StackOverflow - <http://stackoverflow.com/questions/tagged/openacc>
- OpenACC Toolkit - <http://developer.nvidia.com/openacc-toolkit>
- Parallel Forall Blog - <http://devblogs.nvidia.com/parallelforall/>
- GPU Technology Conference - <http://www.gputechconf.com/>
- OpenACC Website - <http://openacc.org/>

Questions? Email openacc@nvidia.com