OpenACC Pipelining

Jeff Larkin <jlarkin@nvidia.com>, November 18, 2016

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

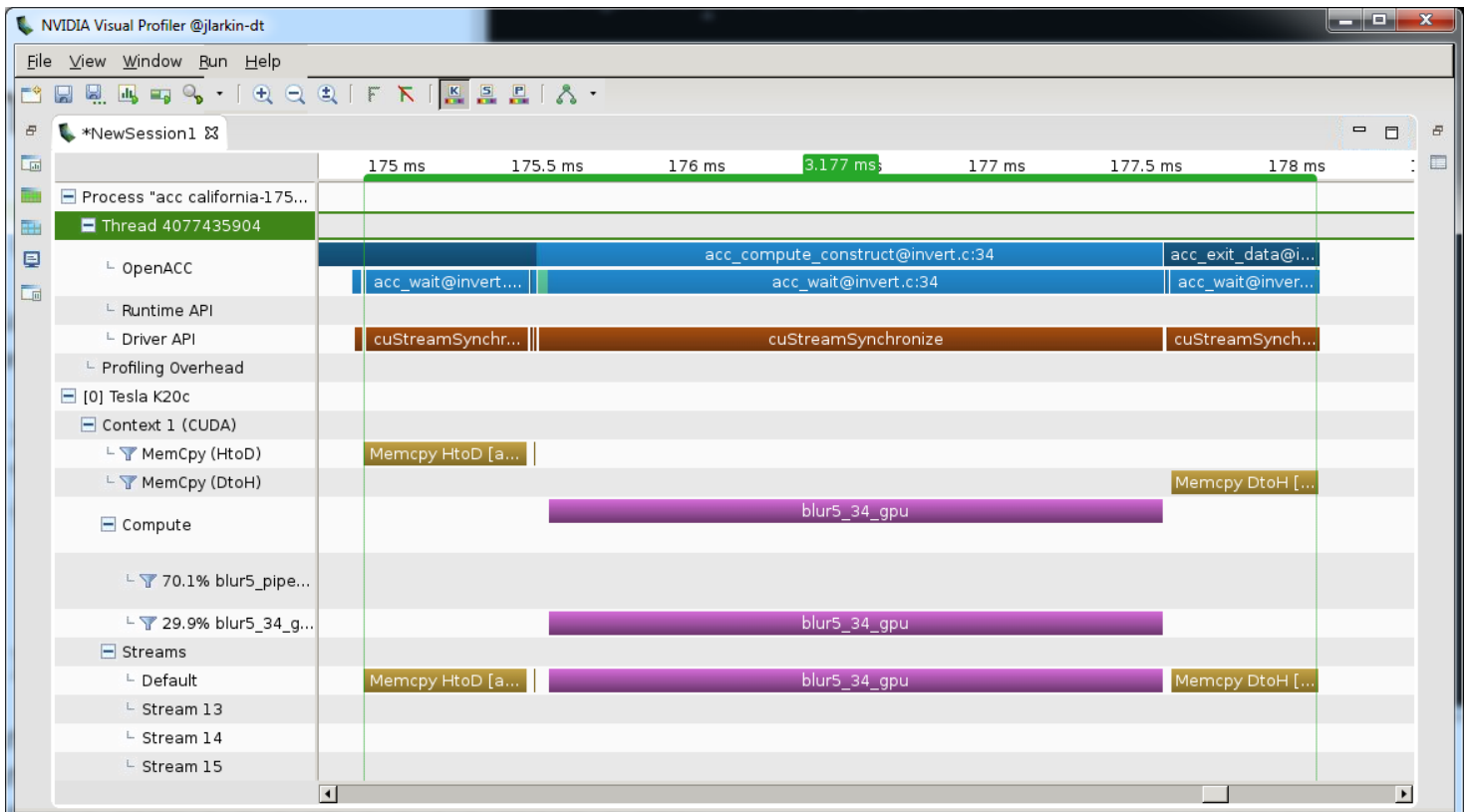
```
#pragma acc parallel loop collapse(2) gang vector
for ( y = 0; y < h; y++ ); for ( x = 0; x < w; x++ )
{
  float blue = 0.0, green = 0.0, red = 0.0;
  for ( int fy = 0; fy < filtersize; fy++ )
  {
    long iy = y - (filtersize/2) + fy;
    for ( int fx = 0; fx < filtersize; fx++ )
    {
      blue+=filter[fy][fx]*imgData[iy*step+ix*ch];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

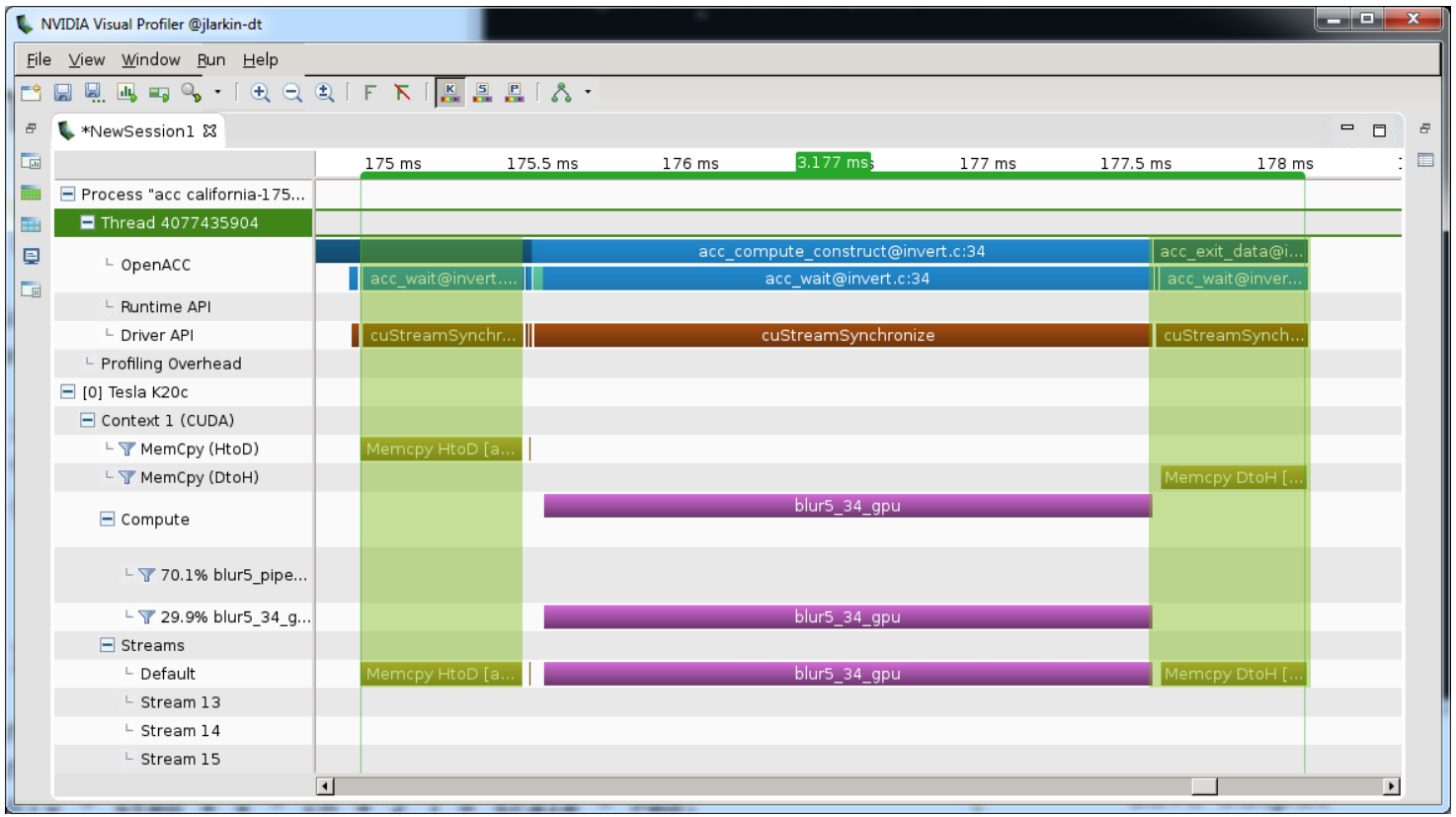
Apply filter

Save output

GPU Timeline

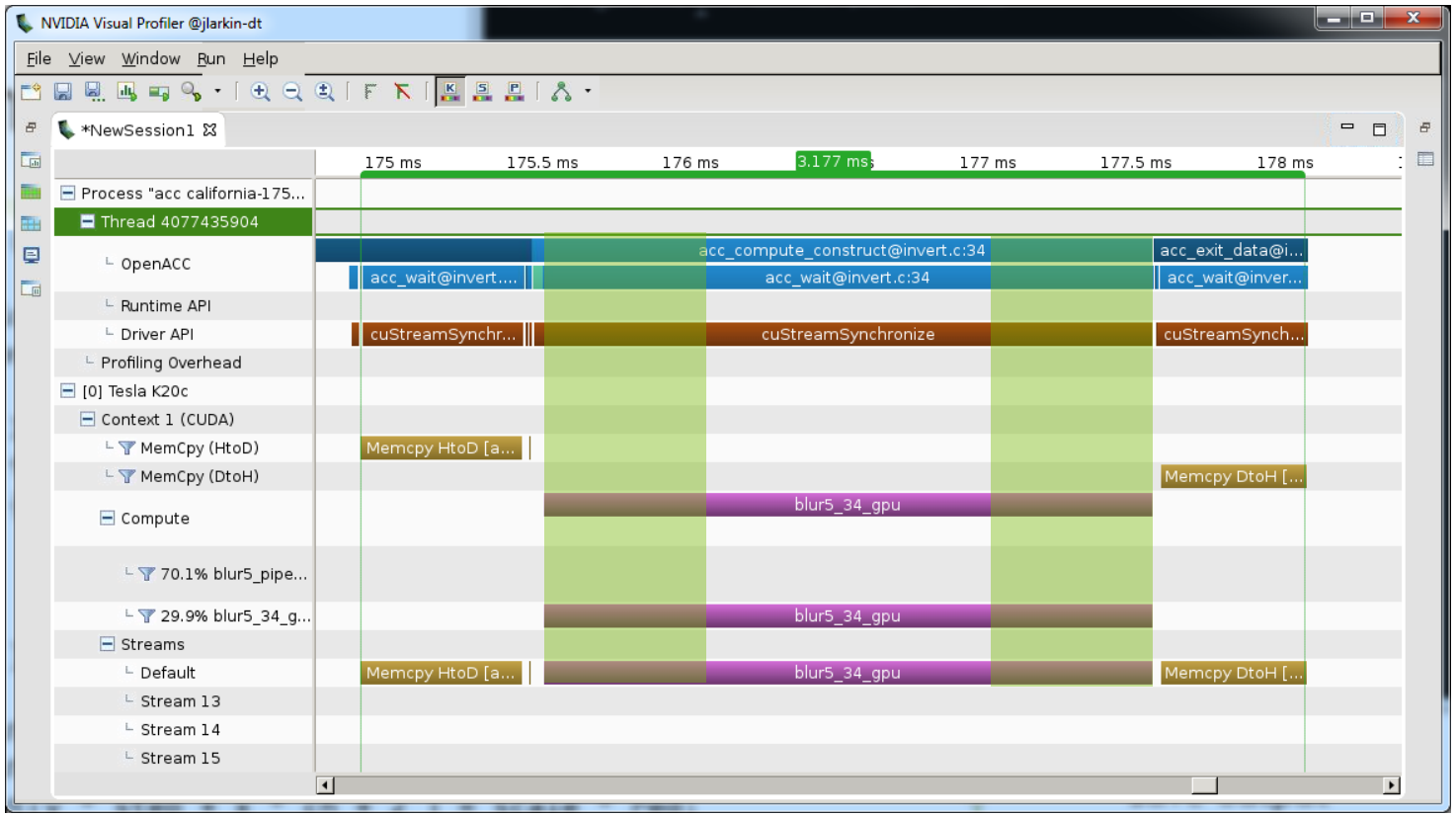


GPU Timeline



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

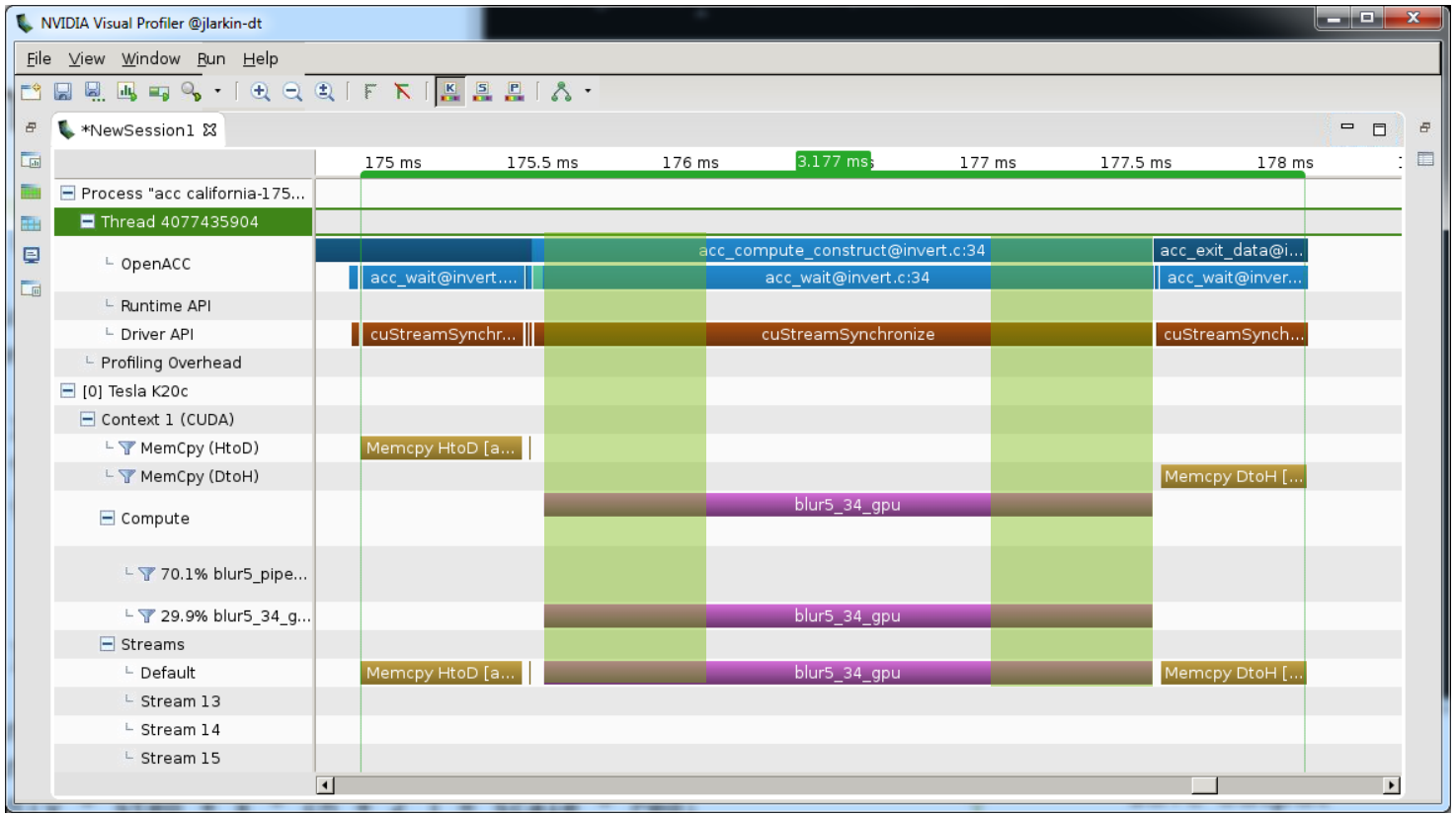
GPU Timeline



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.

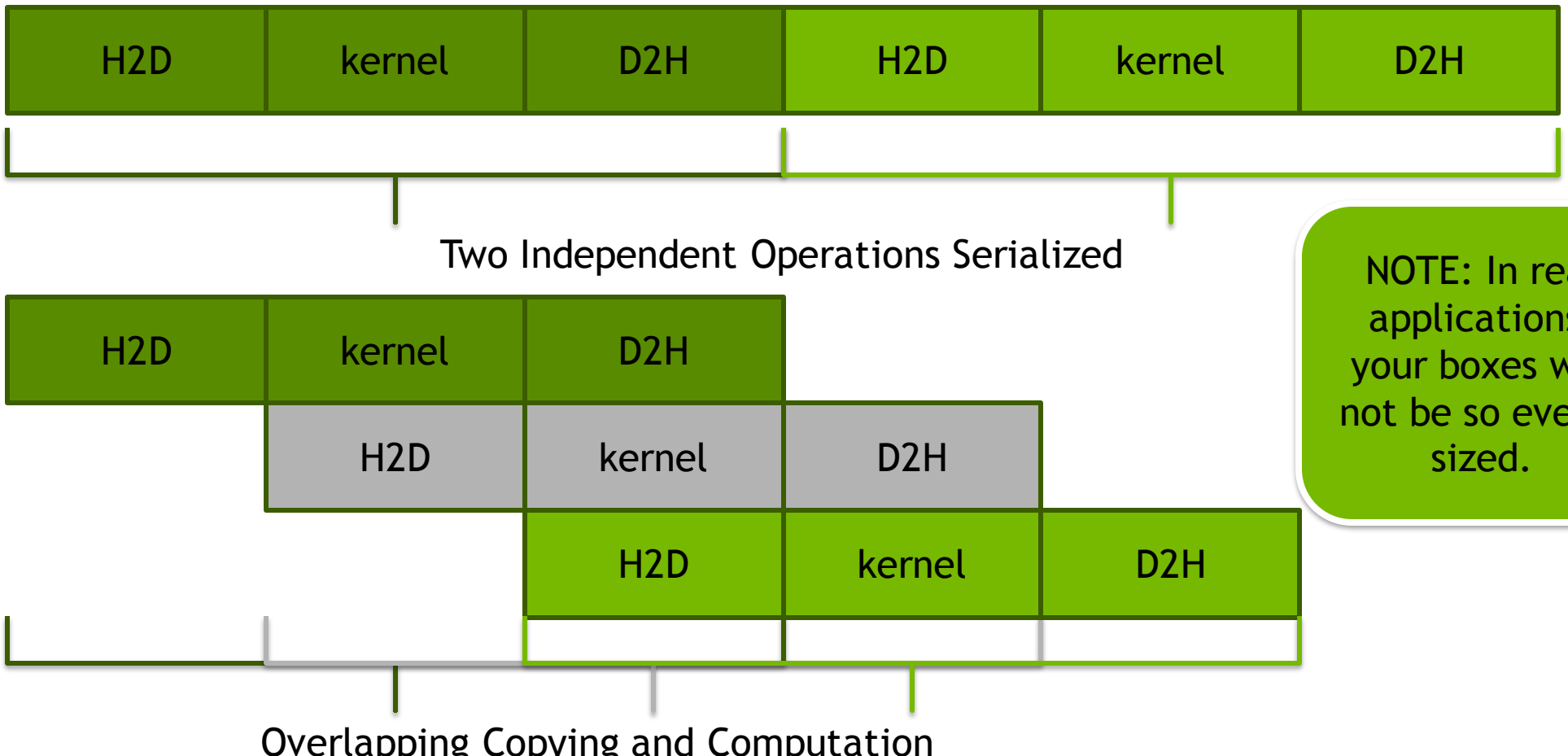
What if we could overlap the copies with the computation?

<u>Rough Math:</u>

3.2ms - .5ms - .5ms = 2.2 ms

3.2 / 2.2 ~= 1.45X

Pipelining Data Transfers



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

NVIDIA.

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

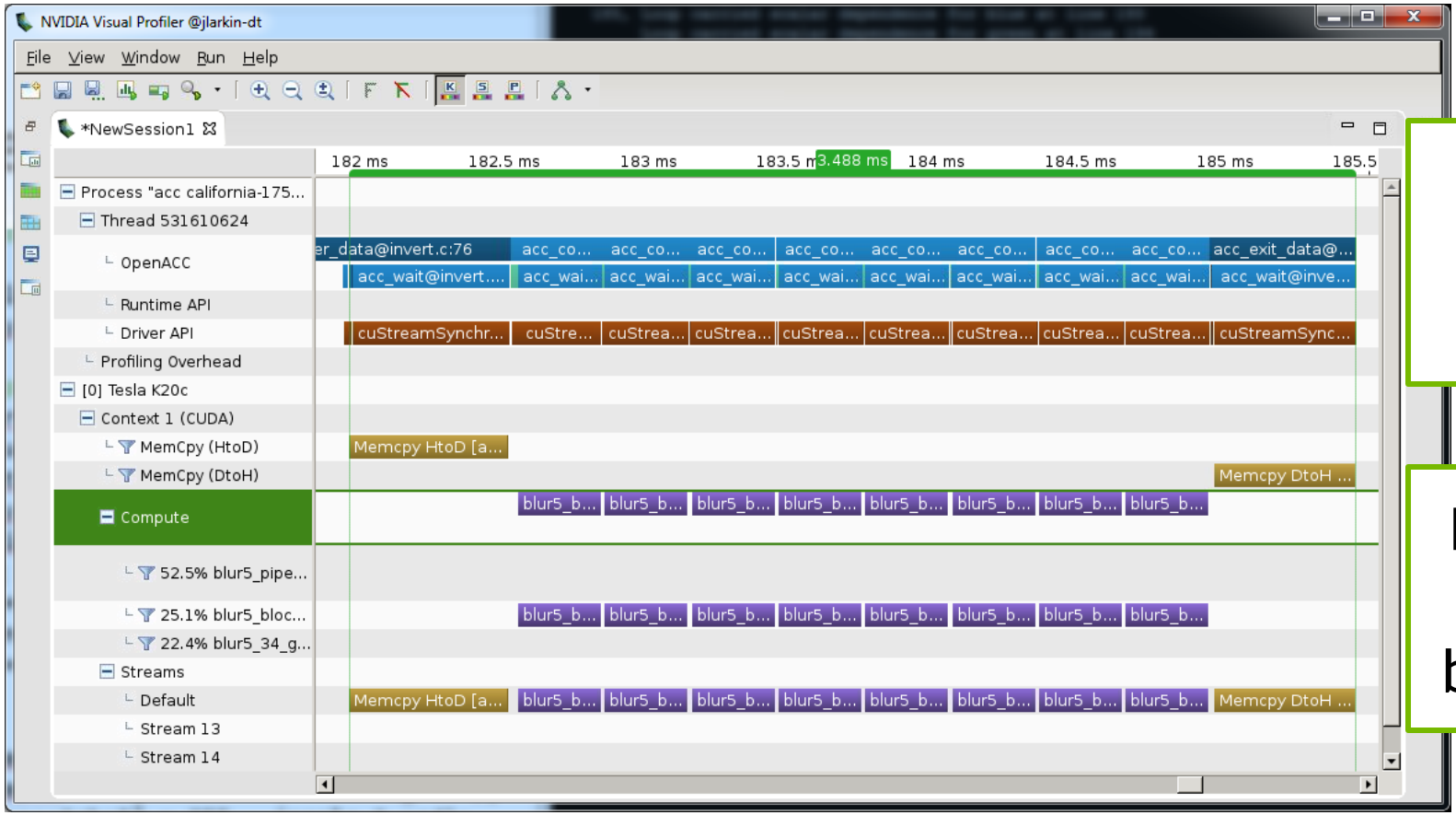
```
#pragma acc data copyin(imgData[:w*h*ch],filter)
                 copyout(out[:w*h*ch])
for ( long blocky = 0; blocky < nblocks; blocky++){
  long starty = blocky * blocksize;
  long endy   = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector
  for (y=starty; y<endy; y++); for(x=0; 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)

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

Copy “a” from GPU to CPU

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:(endy-starty)*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 ommitted>
    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

```
#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:(endy-starty)*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 ommitted>
    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])
}
```

Change data clauses to create

Update data one block at a time.

Blocked Update Code

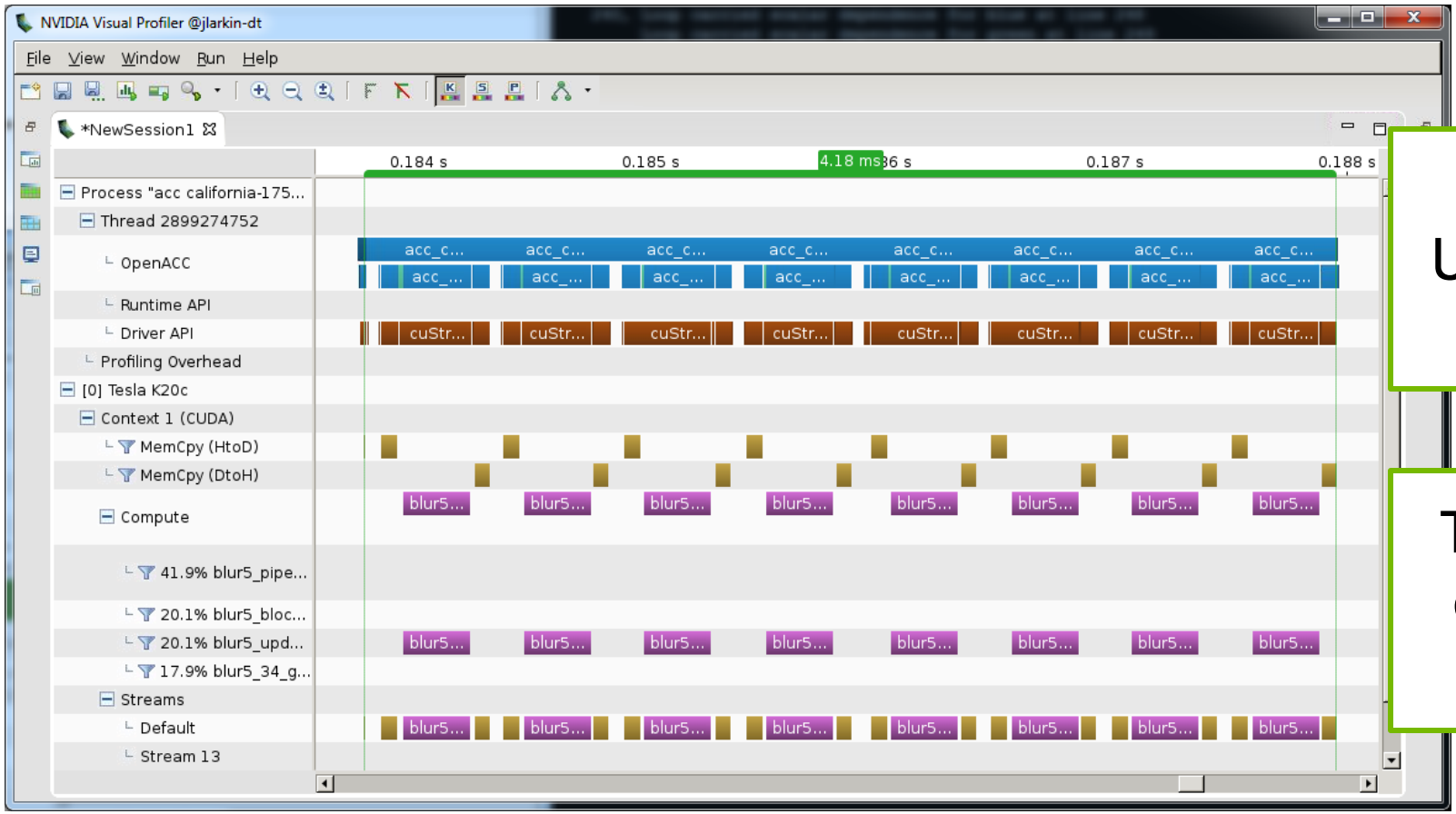
```
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
                copyin(filter)
for ( long blocky = 0; blocky < nblocks; blocky++)
{
  long starty = MAX(0,blocky * blocksize - filtersize/2);
  long endy   = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update device(imgData[starty*step:(endy-starty)*step])
  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 ommitted>
    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])
}
```

Change data clauses to create

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

Work queues operate in-order, serving as a way to express dependencies.

Work queues of different numbers *may* (*or may not*) run concurrently.

```
#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:(endy-starty)*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 ommitted>
    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.

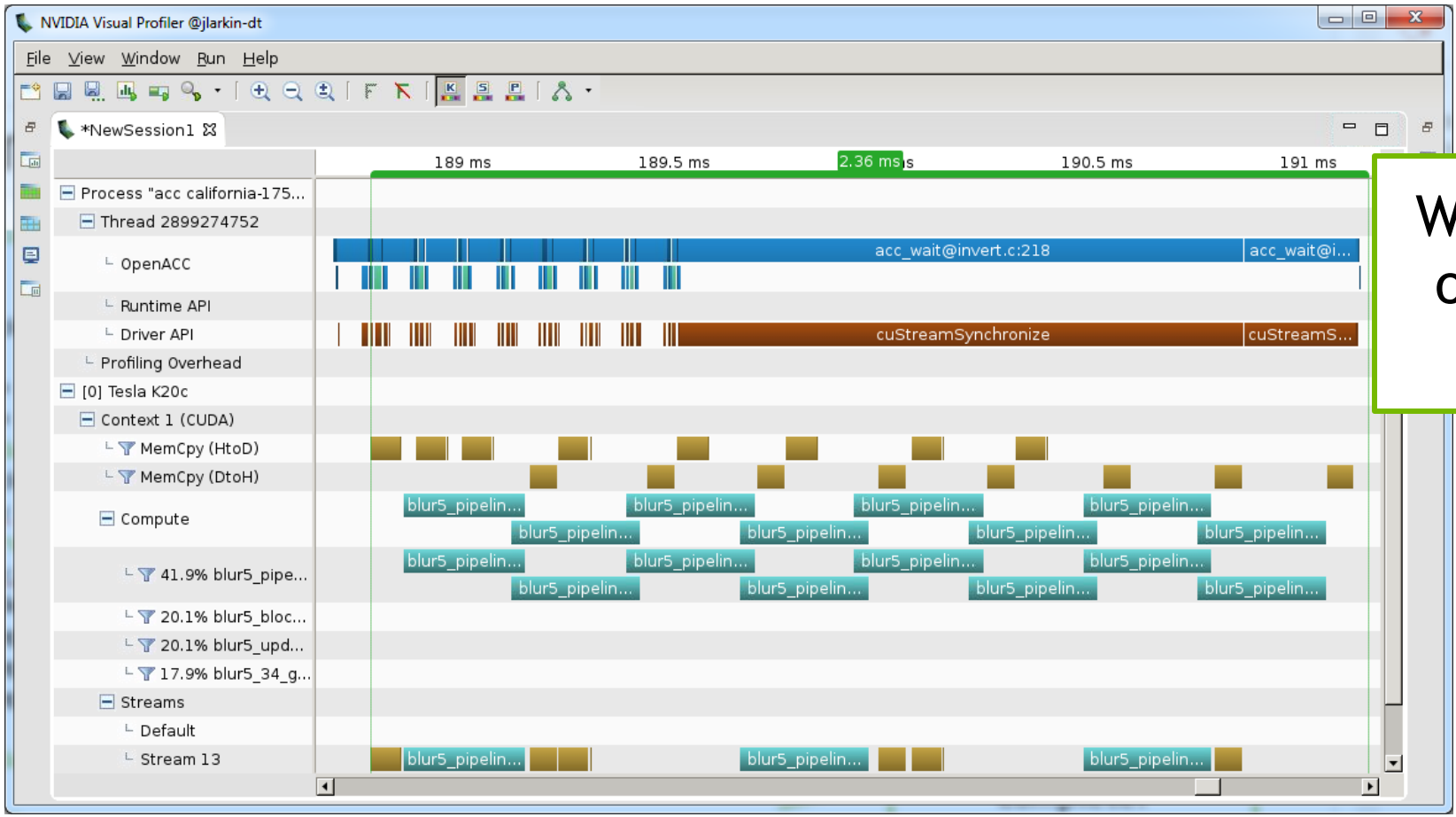
Pipelined Code

```
#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:(endy-starty)*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 ommitted>
    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.

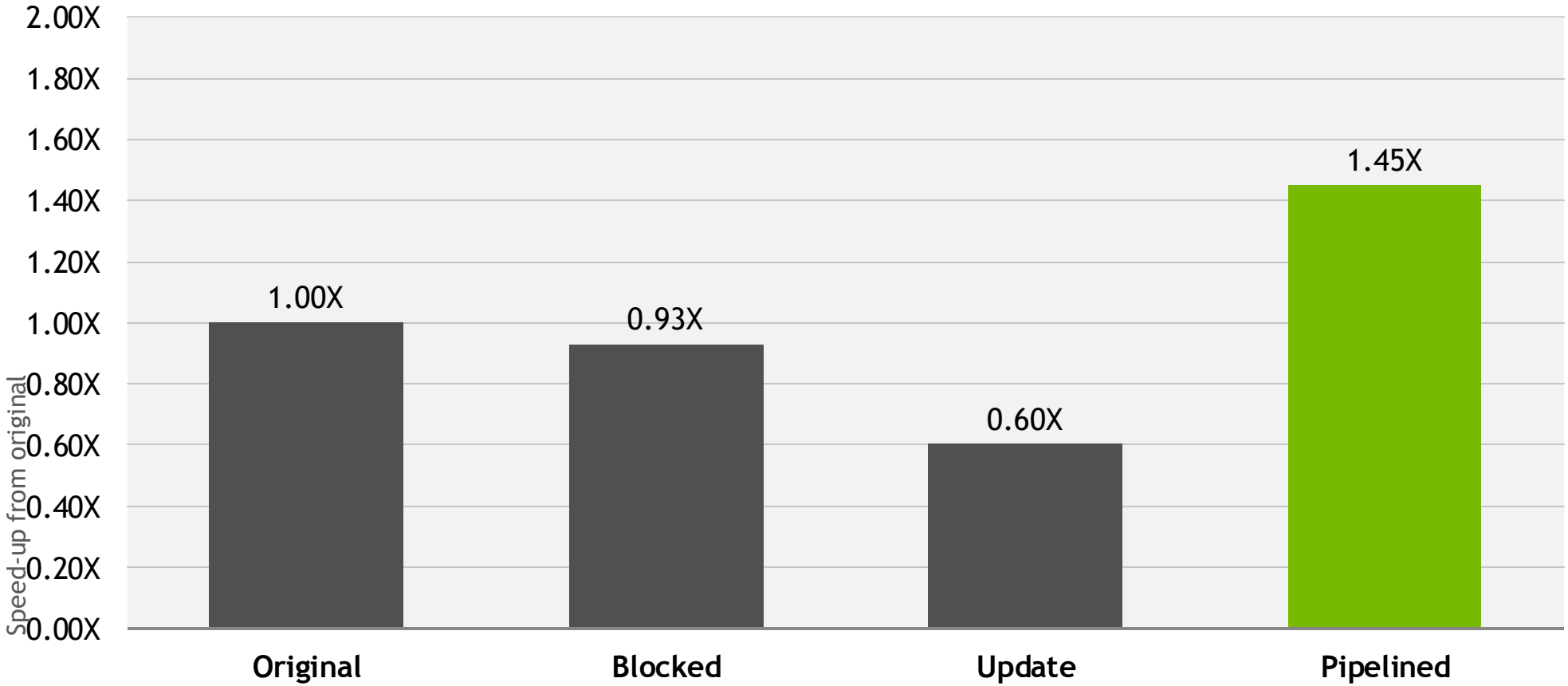
Wait for all blocks to complete.

GPU Timeline Pipelined



We're now able to overlap compute and copy.

Step-by-Step Performance



Source: PGI 16.9, NVIDIA Tesla K20c

Multi-GPU Pipelining

Multi-GPU OpenACC with OpenMP

```
#pragma omp parallel num_threads(acc_get_num_devices(acc_device_nvidia))
  {
    int myid = omp_get_thread_num();
    acc_set_device_num(myid,acc_device_nvidia);
    int queue = 1;
#pragma acc data create(imgData[w*h*ch],out[w*h*ch])
  {
#pragma omp for schedule(static)
  for ( long blocky = 0; blocky < nblocks; blocky++)
  {
    // For data copies we need to include the ghost zones for the filter
    long starty = MAX(0,blocky * blocksize - filtersize/2);
    long endy   = MIN(h,starty + blocksize + filtersize/2);
#pragma acc update device(imgData[starty*step:(endy-starty)*step]) async(queue)
    starty = blocky * blocksize;
    endy = starty + blocksize;
#pragma acc parallel loop collapse(2) gang vector async(queue)
    for ( long y = starty; y < endy; y++ ){ for ( long x = 0; x < w; x++ ){
        <filter code>    } }
#pragma acc update self(out[starty*step:blocksize*step]) async(queue)
    queue = (queue%3)+1;
  }
#pragma acc wait
  }  }
```

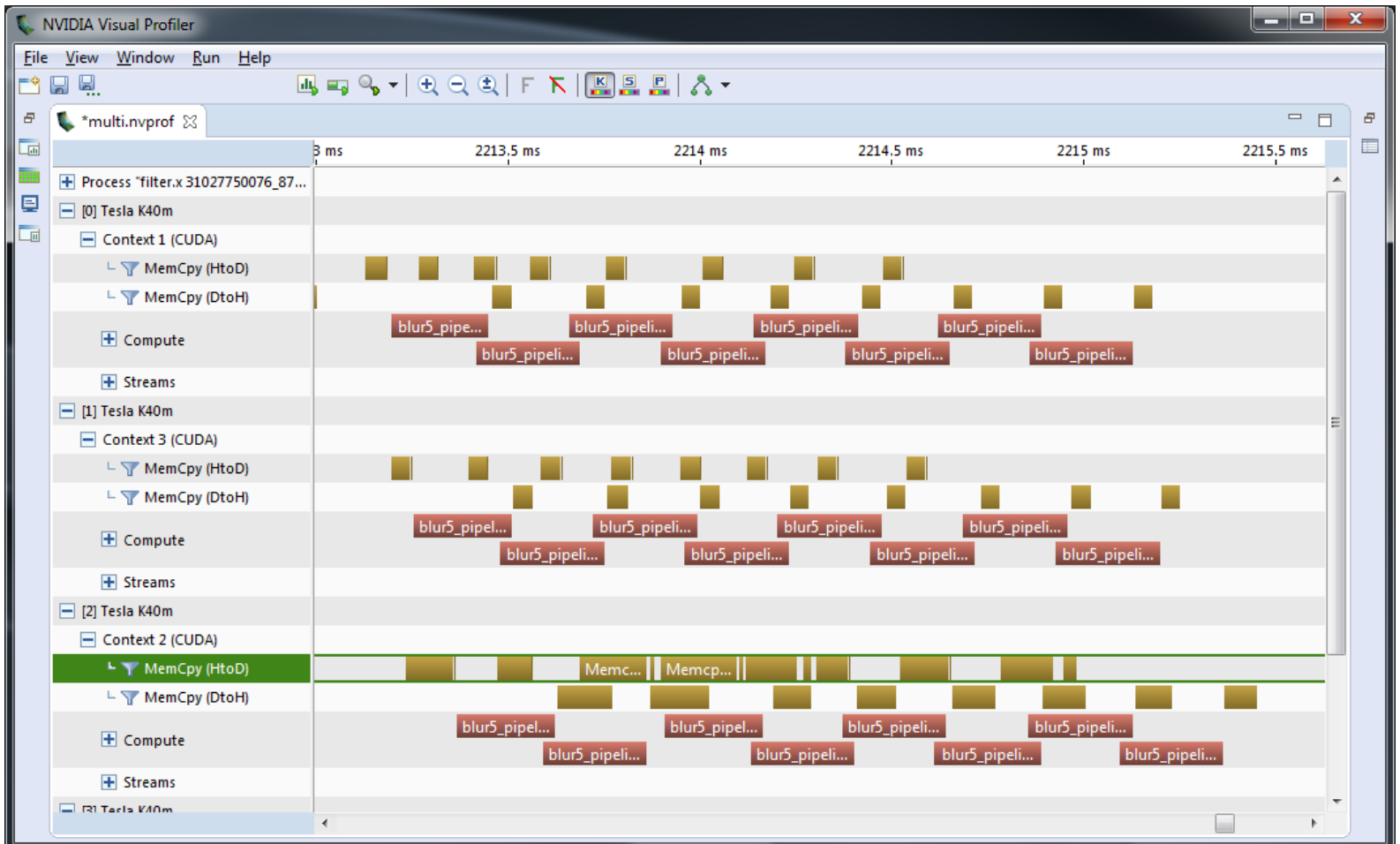
Set the device number, all work will be sent to this device.

Use multiple queues per device to get copy compute overlap

Wait for all work to complete (per device)

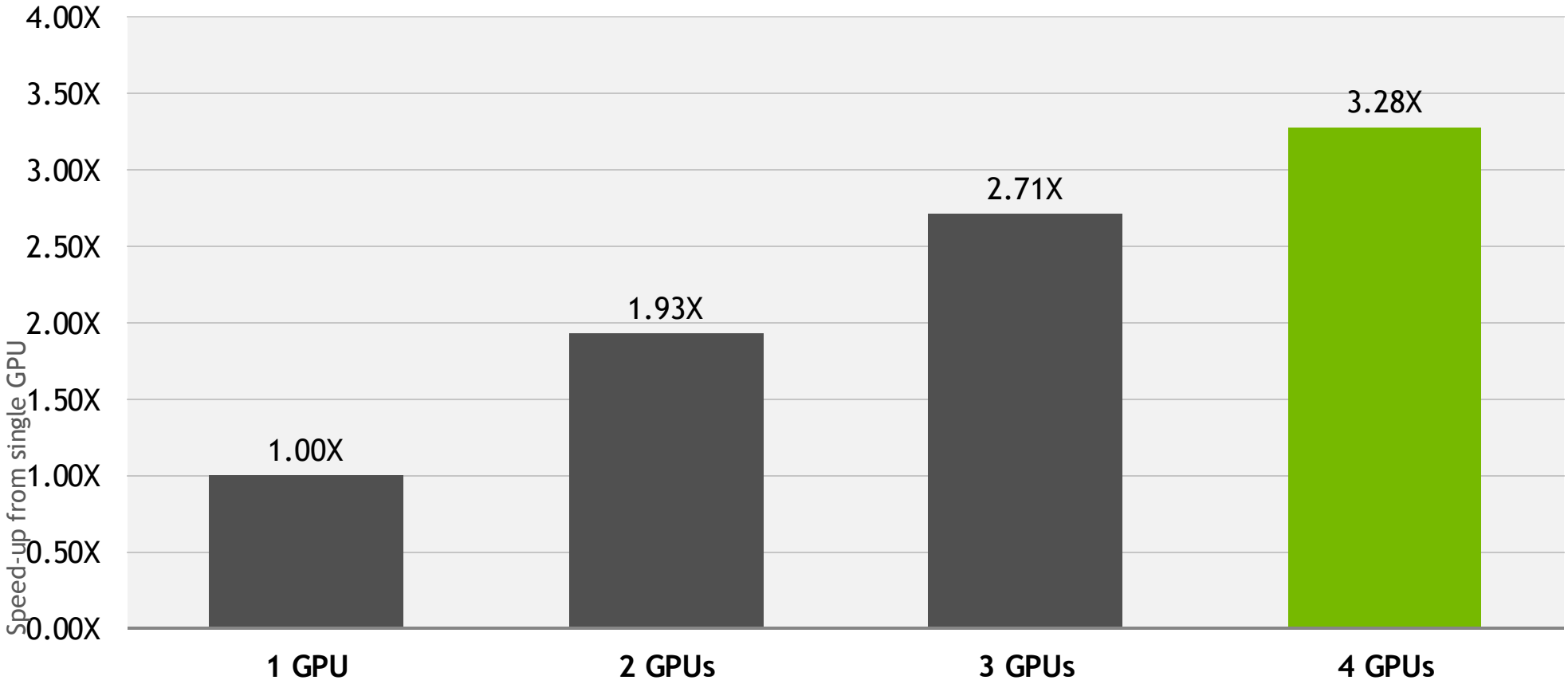
NVIDIA.

Multi-GPU Pipeline Profile



NVIDIA.

Step-by-Step Performance



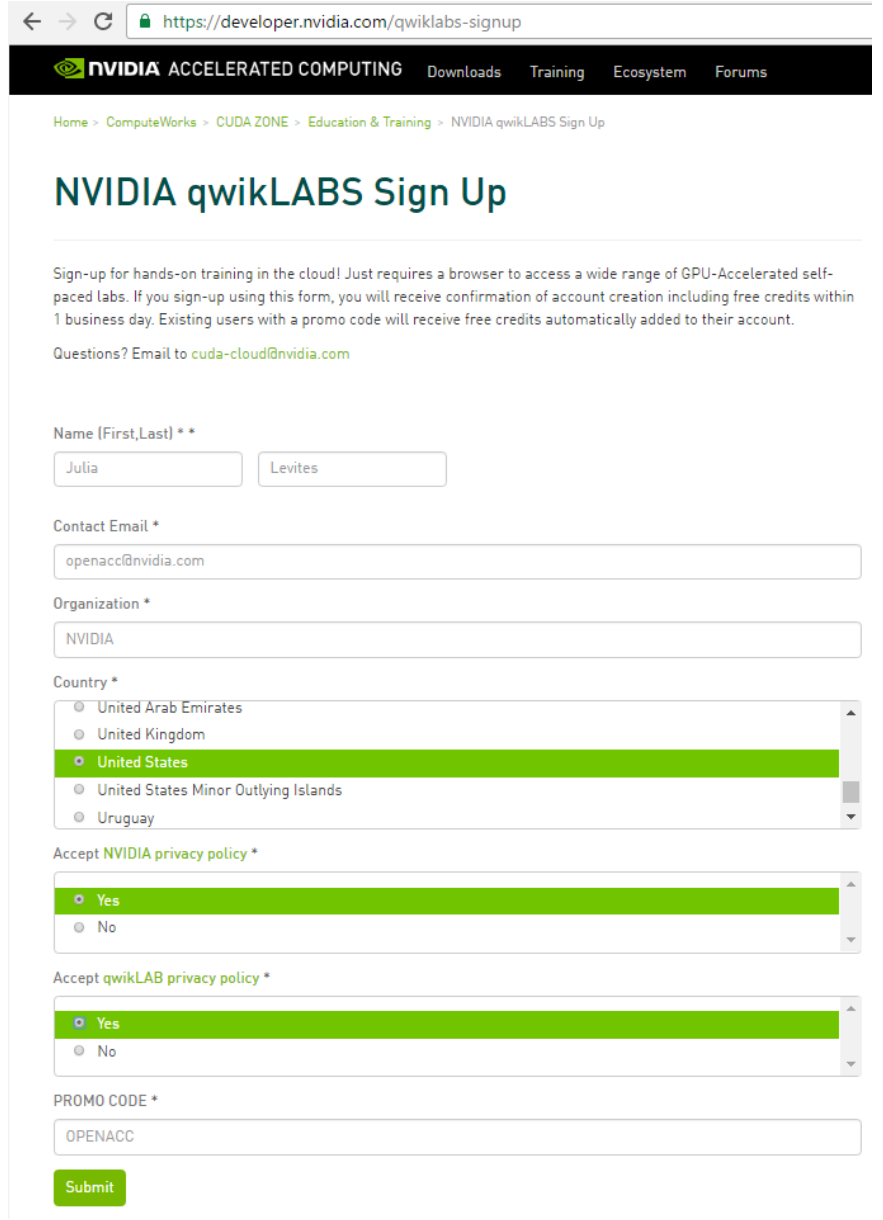
Source: PGI 16.10, NVIDIA Tesla K40

Where to find OpenACC help

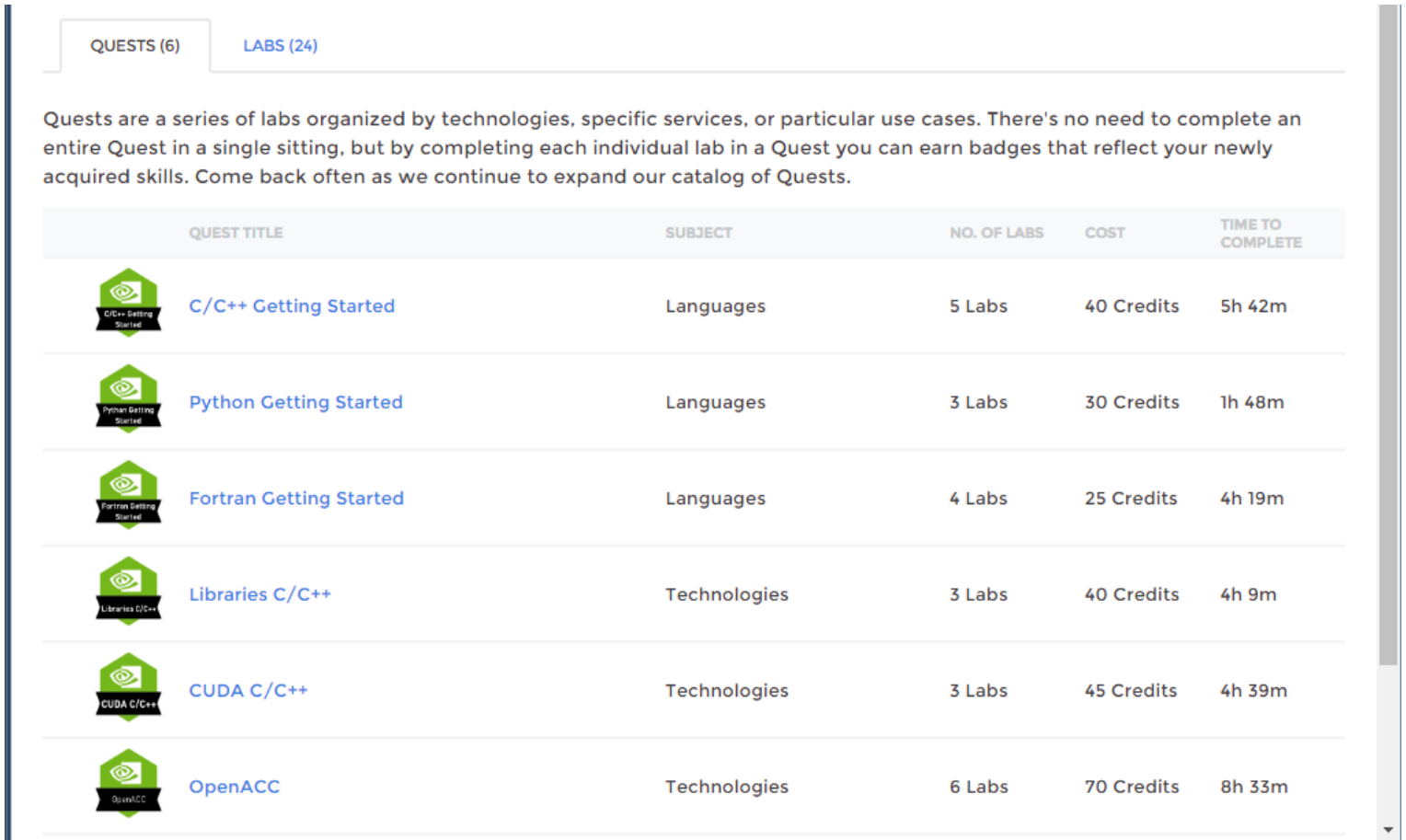
- OpenACC Course Recordings - https://developer.nvidia.com/openacc-courses
- PGI Website - http://www.pgroup.com/resources
- OpenACC on StackOverflow - http://stackoverflow.com/questions/tagged/openacc
- OpenACC Toolkit - http://developer.nvidia.com/openacc-toolkit
- Parallel Forall Blog - http://devblogs.nvidia.com/parallelforall/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

Free Qwiklabs

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



Free Qwiklab Quests

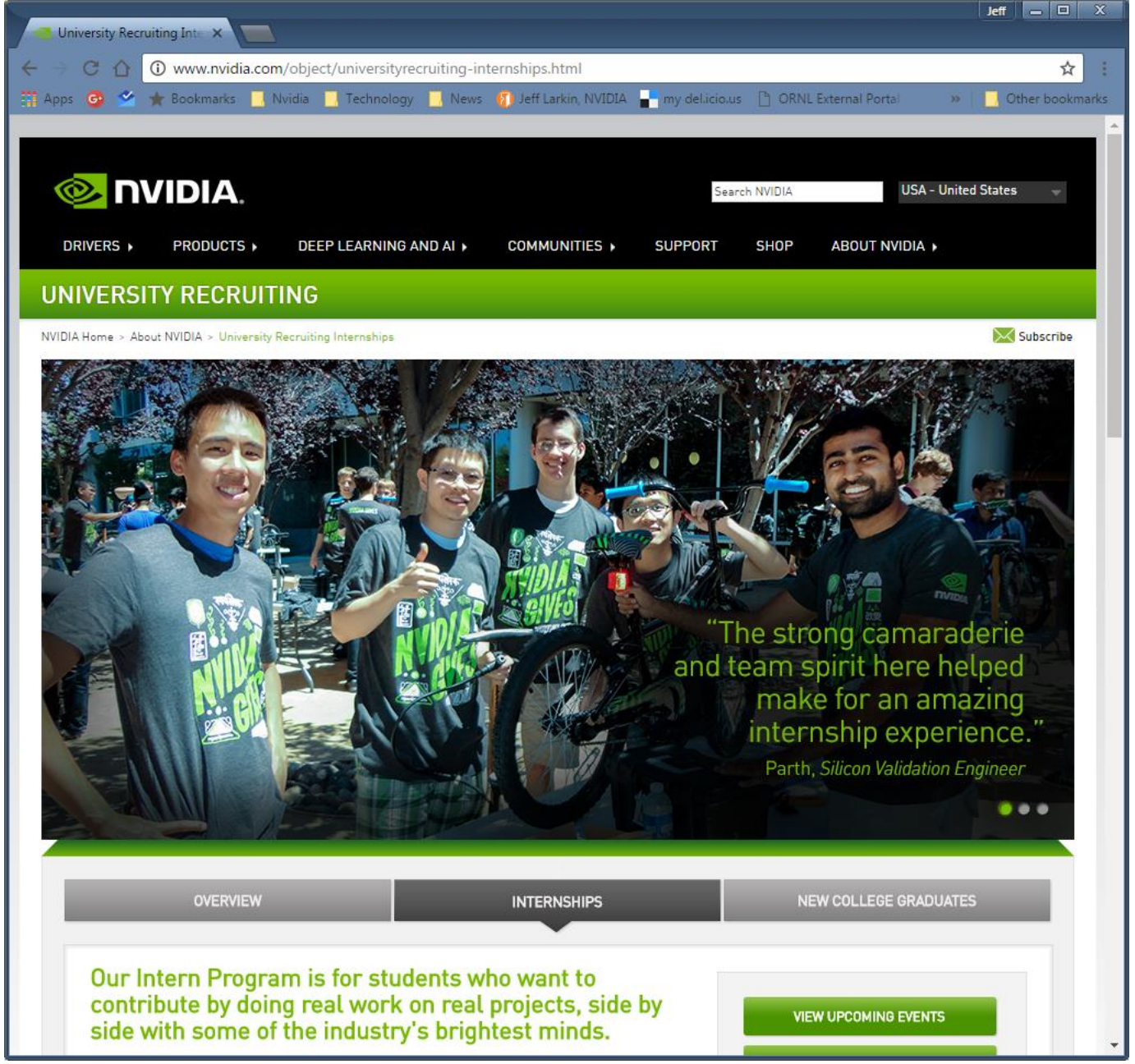


QUESTS (6) LABS (24)

Quests are a series of labs organized by technologies, specific services, or particular use cases. There's no need to complete an entire Quest in a single sitting, but by completing each individual lab in a Quest you can earn badges that reflect your newly acquired skills. Come back often as we continue to expand our catalog of Quests.

QUEST TITLE	SUBJECT	NO. OF LABS	COST	TIME TO COMPLETE
C/C++ Getting Started	Languages	5 Labs	40 Credits	5h 42m
Python Getting Started	Languages	3 Labs	30 Credits	1h 48m
Fortran Getting Started	Languages	4 Labs	25 Credits	4h 19m
Libraries C/C++	Technologies	3 Labs	40 Credits	4h 9m
CUDA C/C++	Technologies	3 Labs	45 Credits	4h 39m
OpenACC	Technologies	6 Labs	70 Credits	8h 33m

Email me if you run out of credits, I can always get you more!



Looking for a summer internship? Go to www.nvidia.com, look for Careers at the bottom of the page.