A progression of OpenCL exercises

Tim Mattson, Intel

Disclaimer

- **I am speaking for myself and not my employer (Intel).**
- **I am not making any claims about Intel products or performance you might achieve with Intel products.**
  - I work in a research lab and know nothing about Intel products that you couldn’t find from online sources.

Agenda for the afternoon

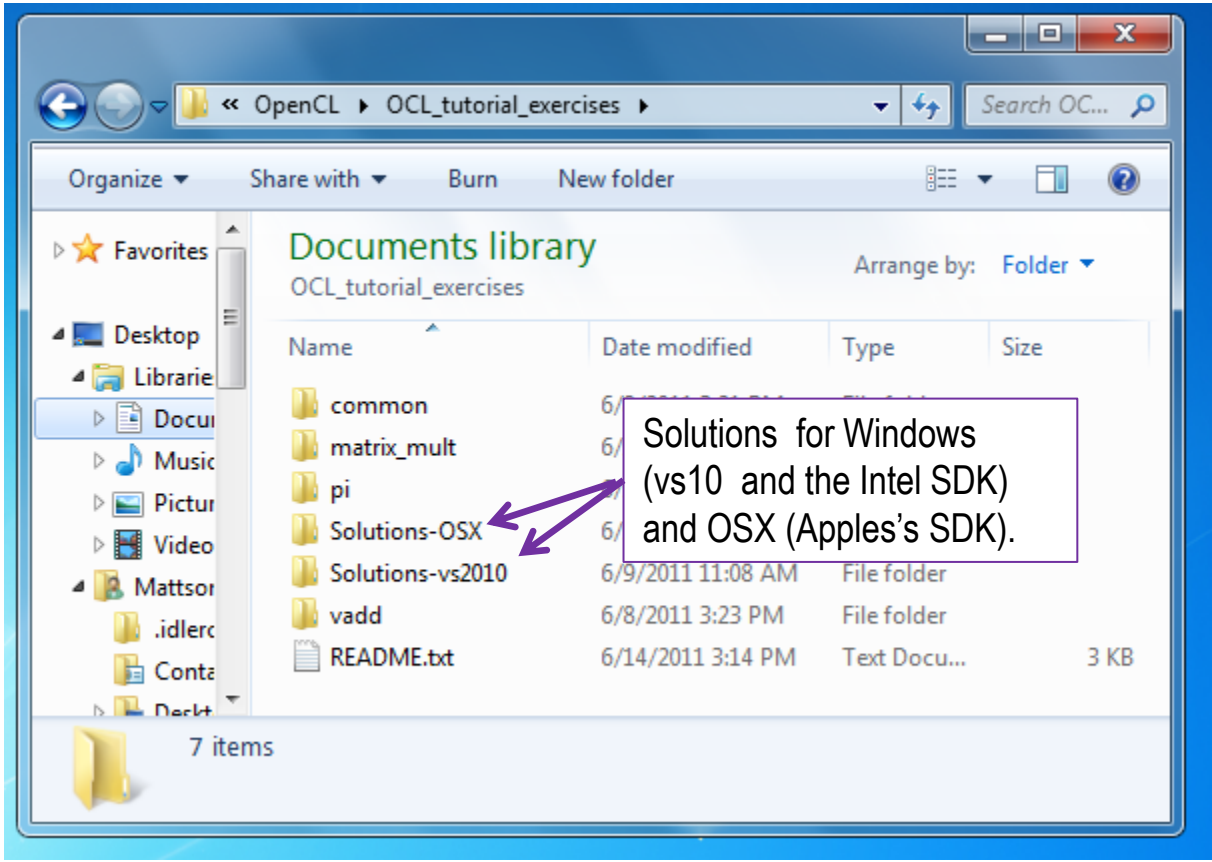
- **You will gain experience**
  - Building simple OpenCL programs.
  - Working with the OpenCL memory model
  - Using the Event model in OpenCL

- **We will NOT cover**
  - Installing OpenCL (I assume you've done that already).
  - OpenCL Comparisons: Apple vs. Intel vs. AMD vs. NVIDIA
  - Benchmarking

Our Goal today is pedagogy … To make you comfortable writing basic OpenCL programs.

Assumptions

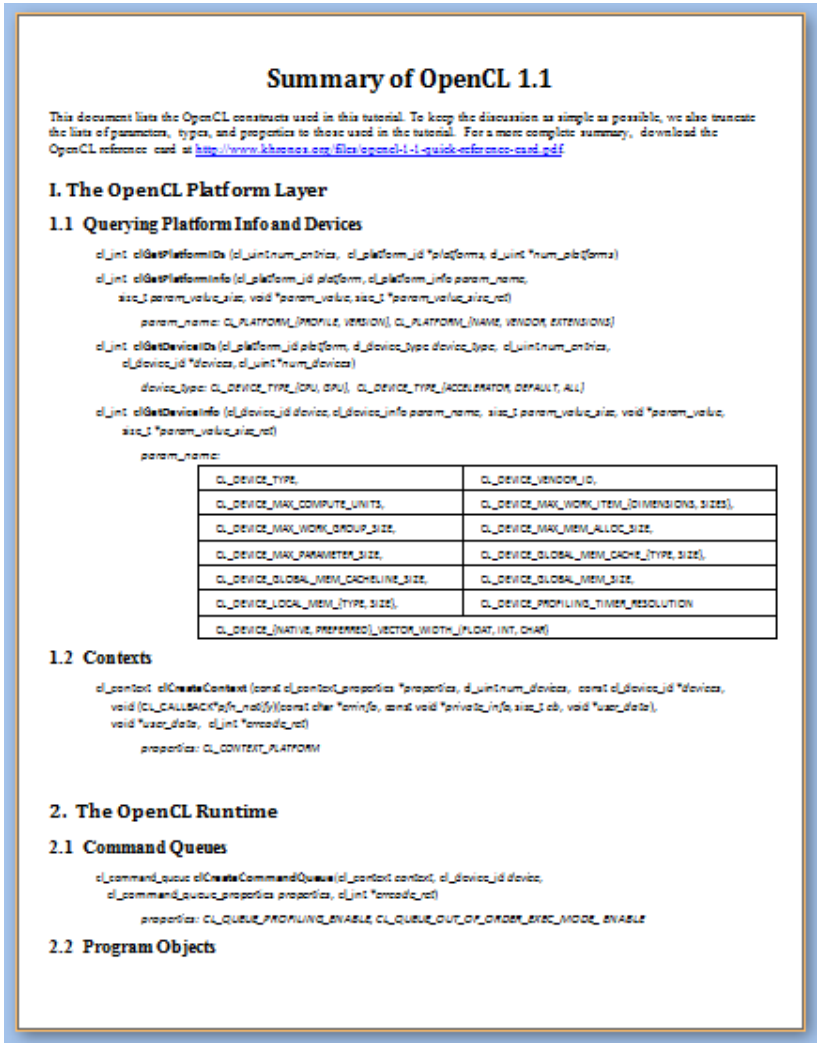
- **I assume the following:**
  - You have a working implementation of OpenCL, either on your laptop of on a Linux server you can reach from you laptop.
  - You have read the documentation on how to use your implementation of OpenCL (i.e. we can't spend time figuring out AMD vs. Apple vs. Intel vs. Nvidia).



- **Do not cheat by looking at the solutions. For effective learning, you must solve these problems on your own.**

Summary of OpenCL API

- **OpenCL is huge. Fortunately, for most programs you use only a small subset of OpenCL. I have provided a summary of this subset in a 4 page handout.**



Summary of OpenCL 1.1

This document lists the OpenCL constructs used in this tutorial. To keep the discussion as simple as possible, we also truncate the lists of parameters, types, and properties to those used in the tutorial. For a more complete summary, download the OpenCL reference card at http://www.khronos.org/files/opencl-1-1-quick-reference-card.pdf

I. The OpenCL Platform Layer

1.1 Querying Platform Info and Devices

cl_int **clGetPlatformIDs** (cl_uint num_entries, cl_platform_id *platforms, cl_uint *num_platforms)

cl_int **clGetPlatformInfo** (cl_platform_id platform, cl_platform_info param_name,
size_t param_value_size, void *param_value, size_t *param_value_size_ret)

param_name: CL_PLATFORM_{PROFILE, VERSION}, CL_PLATFORM_{NAME, VENDOR, EXTENSIONS}

cl_int **clGetDeviceIDs** (cl_platform_id platform, cl_device_type device_type, cl_uint num_entries,
cl_device_id *devices, cl_uint *num_devices)

device_type: CL_DEVICE_TYPE_{CPU, GPU}, CL_DEVICE_TYPE_{ACCELERATOR, DEFAULT, ALL}

cl_int **clGetDeviceInfo** (cl_device_id device, cl_device_info param_name, size_t param_value_size, void *param_value,
size_t *param_value_size_ret)

param_name:

CL_DEVICE_TYPE,	CL_DEVICE_VENDOR_ID,
CL_DEVICE_MAX_COMPUTE_UNITS,	CL_DEVICE_MAX_WORK_ITEM_{DIMENSIONS, SIZES},
CL_DEVICE_MAX_WORK_GROUP_SIZE,	CL_DEVICE_MAX_MEM_ALLOC_SIZE,
CL_DEVICE_MAX_PARAMETER_SIZE,	CL_DEVICE_GLOBAL_MEM_CACHE_{TYPE, SIZE},
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE,	CL_DEVICE_GLOBAL_MEM_SIZE,
CL_DEVICE_LOCAL_MEM_{TYPE, SIZE},	CL_DEVICE_PROFILING_TIMER_RESOLUTION
CL_DEVICE_{NATIVE, PREFERRED}_VECTOR_WIDTH_{FLOAT, INT, CHAR}	

1.2 Contexts

cl_context **clCreateContext** (const cl_context_properties *properties, cl_uint num_devices, const cl_device_id *devices,
void (CL_CALLBACK*pfn_notify)(const char *errinfo, const void *private_info, size_t cb, void *user_data),
void *user_data, cl_int *errcode_ret)

properties: CL_CONTEXT_PLATFORM

2. The OpenCL Runtime

2.1 Command Queues

cl_command_queue **clCreateCommandQueue**(cl_context context, cl_device_id device,
cl_command_queue_properties properties, cl_int *errcode_ret)

properties: CL_QUEUE_PROFILING_ENABLE, CL_QUEUE_OUT_OF_ORDER_EXEC_MODE_ENABLE

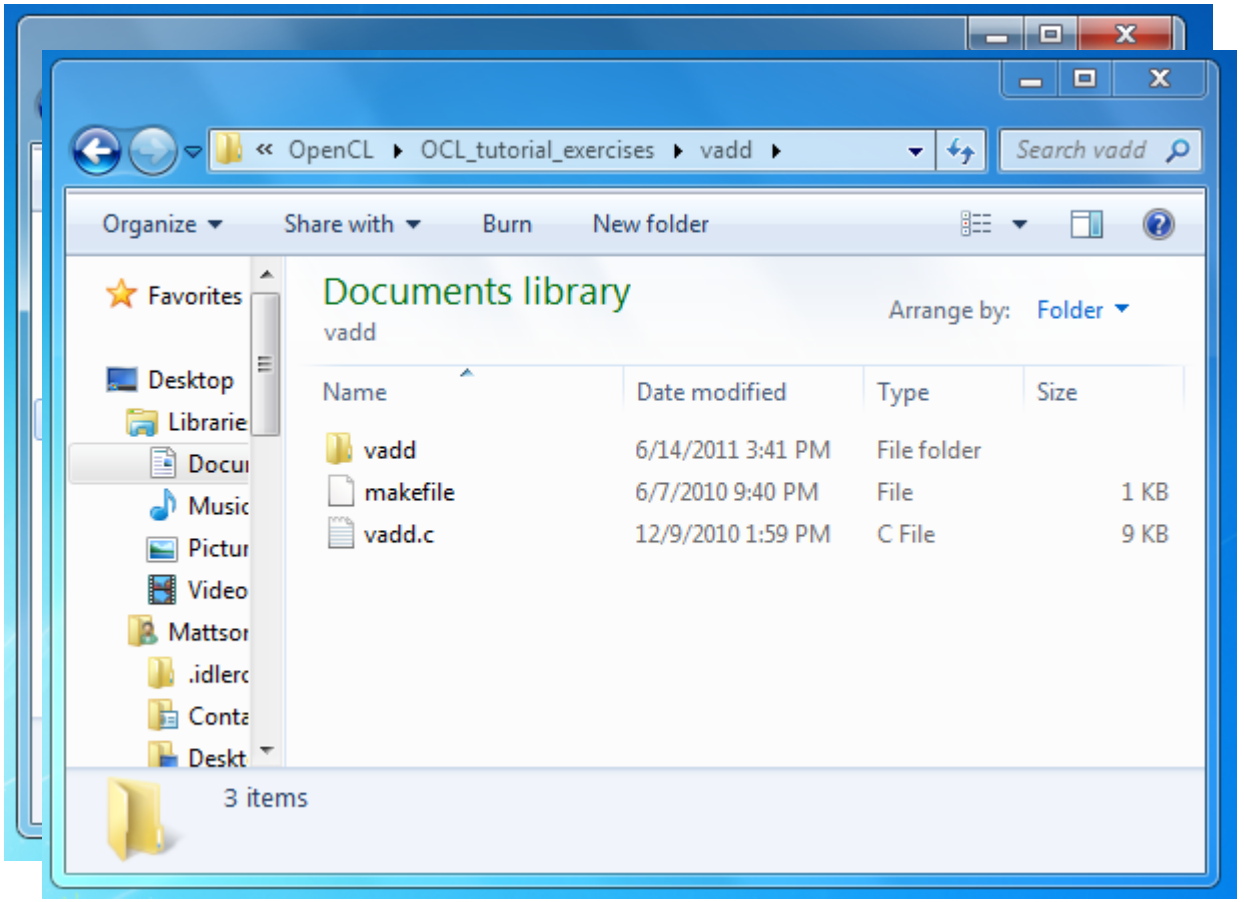
2.2 Program Objects

Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

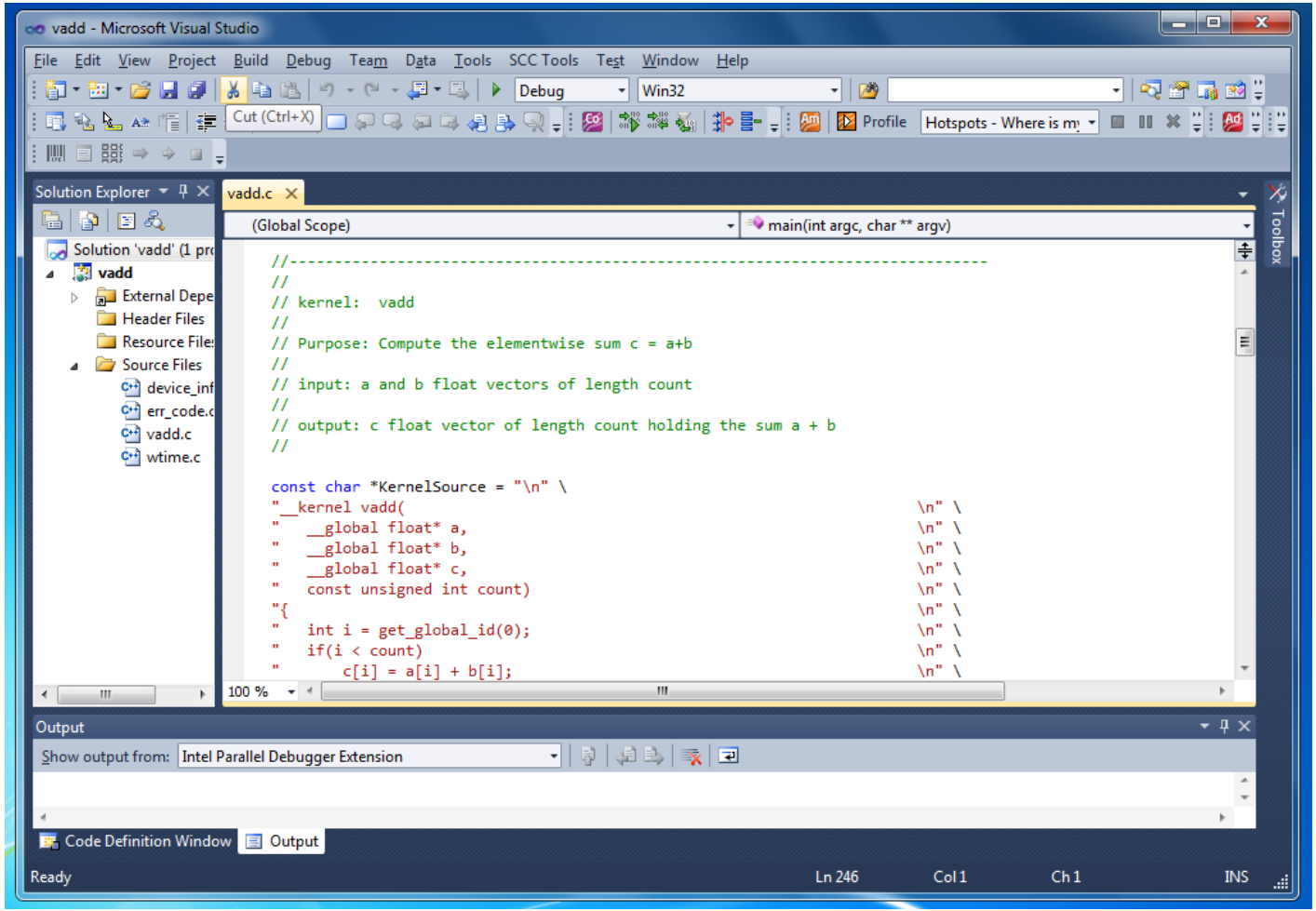
Building and running an OpenCL program

- **Go to the provided vadd folder**
  - On OSX or Linux, modify the make file to support your local OpenCL implementation, type make, then run the produced executable.
  - On Windows using the Intel OpenCL SDK, go to the vadd/vadd folder and double click vadd.sln. Use Build/Rebuild and the Debug/Start-without-debugging menus.



The OpenCL Vadd program

- **Study the source code and ask questions.**

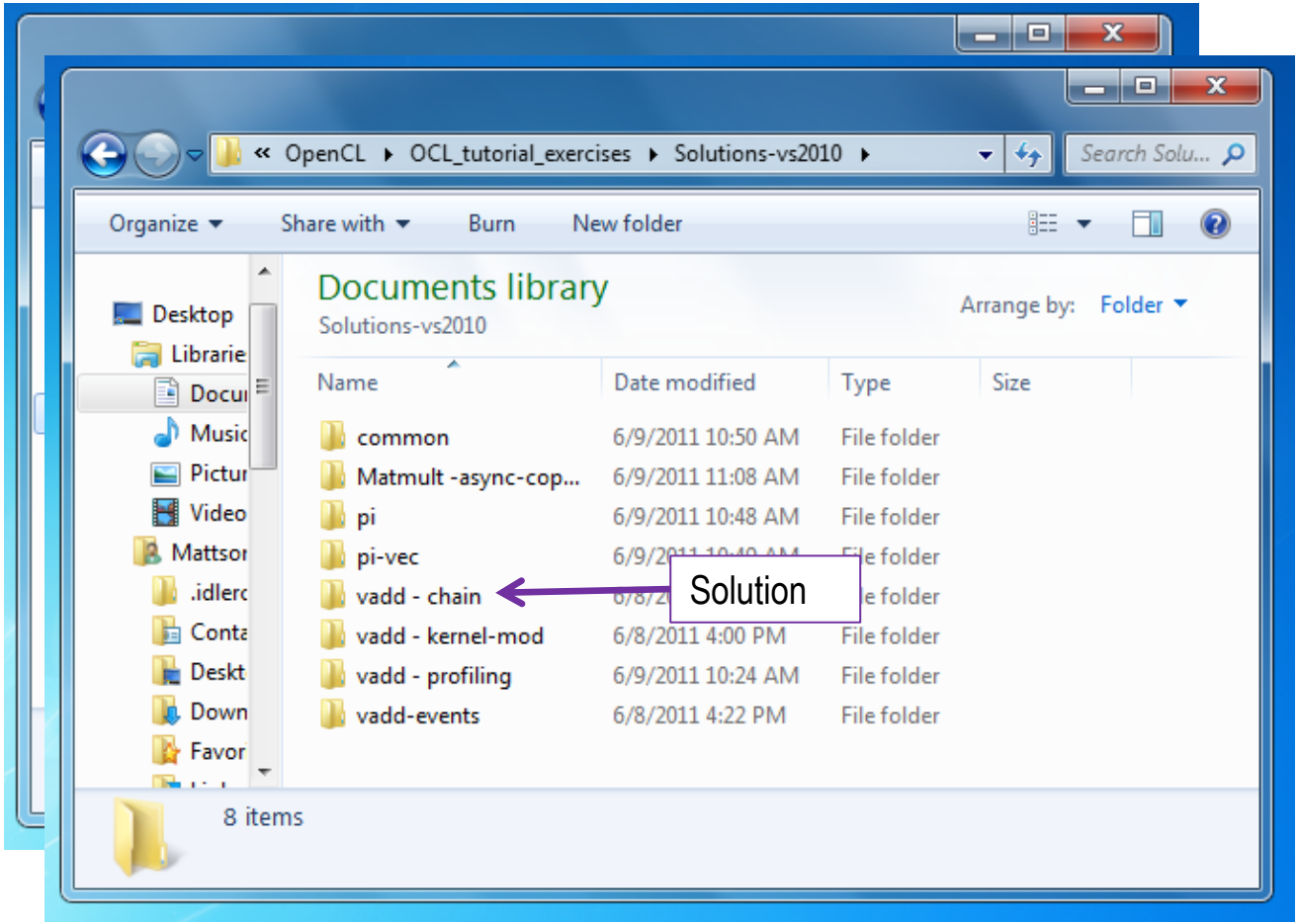


Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Multiple commands in a queue

- **Go to the provided vadd folder**
  - Modify the vadd program to apply the vadd kernel multiple times:
    - C = A + B
    - D = C + A
    - E = D + B

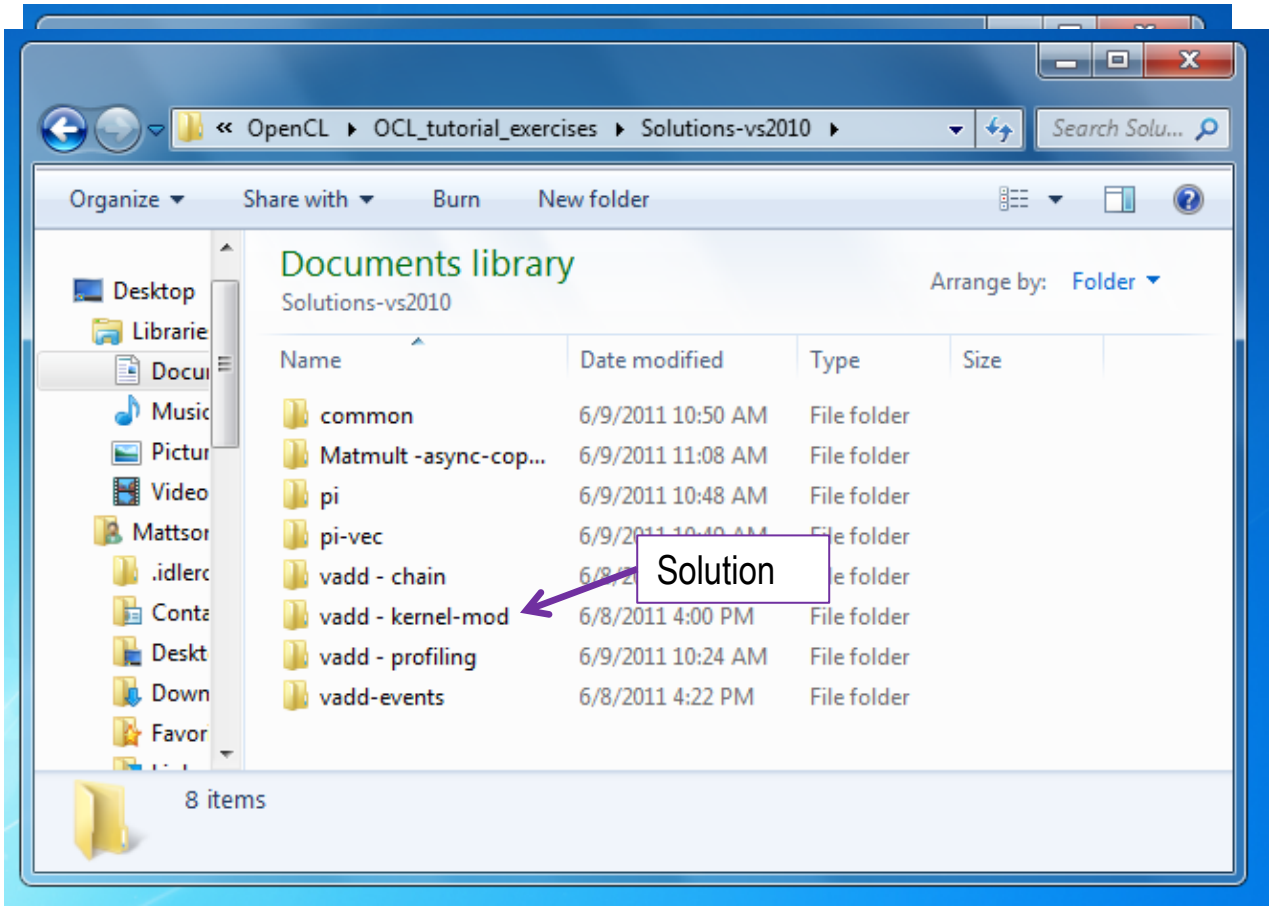


Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Modifying a kernel

- **Go to the vadd folder**
  - Create a new kernel that adds three vectors together
    - D = A + B + C
- Incorporate it with your "chain of vadds" from the previous exercise

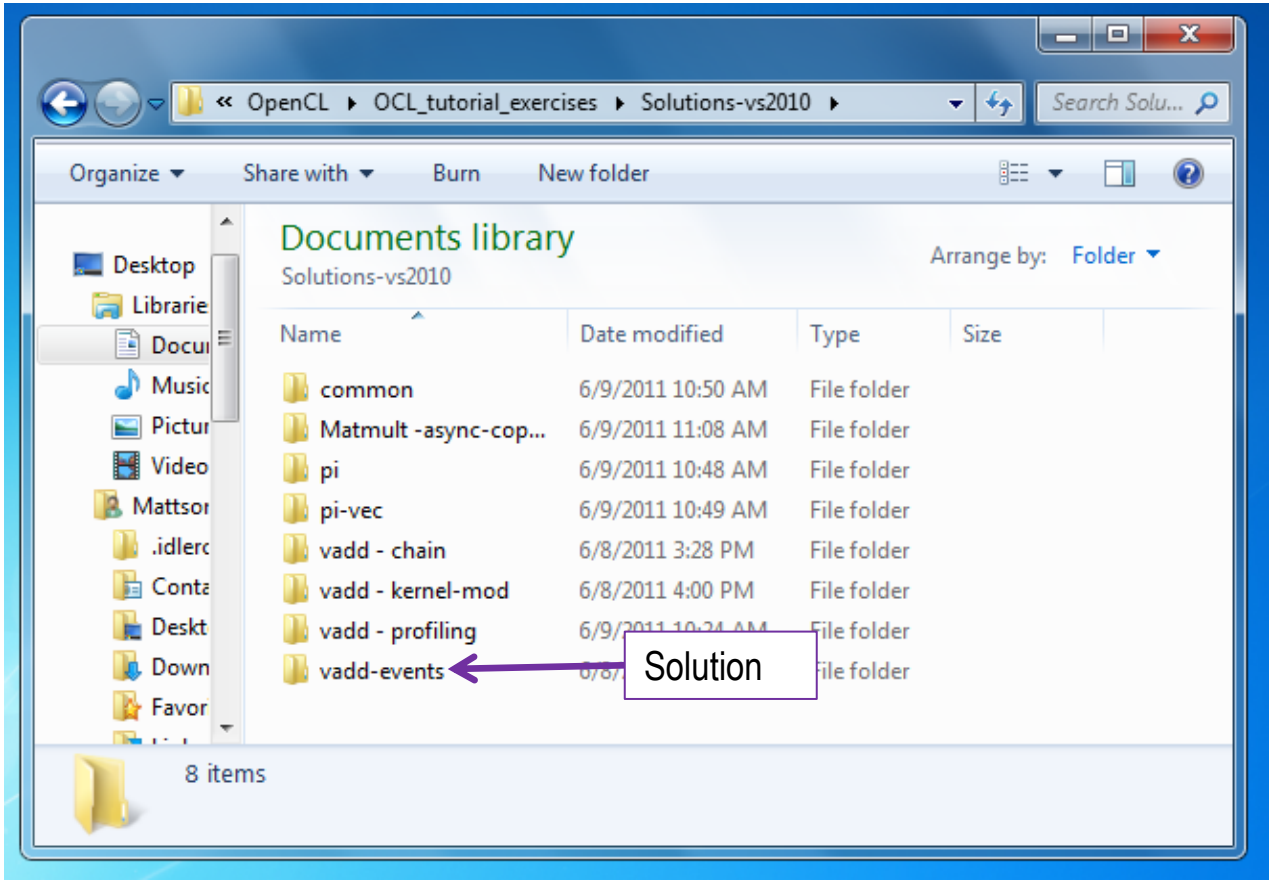


Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Events and out of order Queues

- **Work with your vadd with multiple kernels chained together**
  - Make the queue an out of order queue
  - Add events to the different vadd kernel instances to they satisfy order constraints.



Events

- **An event is an object that communicates the status of commands in OpenCL … legal values for an event:**
  - **CL_QUEUED**: command has been enqueued.
  - **CL_SUBMITED**: command has been submitted to the compute device
  - **CL_RUNNING**: compute device is executing the command
  - **CL_COMPLETE**: command has completed
  - **ERROR_CODE**: a negative value, indicates an error condition occurred.
- **Can query the value of an event from the host … for example to track the progress of a command.**

```
cl_int clGetEventInfo (
        cl_event event,    cl_event_info param_name,
        size_t param_value_size,      void *param_value,
        size_t *param_value_size_ret)
```

- **Examples:**
  - **CL_EVENT_CONTEXT**
  - **CL_EVENT_COMMAND_EXECUTION_STATUS**
  - **CL_EVENT_COMMAND_TYPE**

Generating and consuming events

- Consider the command to enqueue a kernel. The last three arguments optionally expose events (NULL otherwise).

```
cl_int clEnqueueNDRangeKernel (
    cl_command_queue command_queue,
    cl_kernel kernel,     cl_uint work_dim,
    const size_t *global_work_offset,
    const size_t *global_work_size,
    const size_t *local_work_size,
    cl_uint num_events_in_wait_list,
    const cl_event *event_wait_list,
    cl_event *event)
```

- Number of events this command is waiting to complete before executing

- Array of pointers to the events being waited upon … Command queue and events must share a context.

- Pointer to an event object generated by this command.

Event: basic event usage

- Events can be used to impose order constraints on kernel execution.
- Very useful with out of order queues.

```
cl_event    k_events[2];

err = clEnqueueNDRangeKernel(commands, kernel1, 1,
        NULL, &global, &local, 0, NULL, &k_events[0]);

err = clEnqueueNDRangeKernel(commands, kernel2, 1,
        NULL, &global, &local, 0, NULL, &k_events[1]);

err = clEnqueueNDRangeKernel(commands, kernel3, 1,
        NULL, &global, &local, 2, k_events, NULL);
```

- Enqueue two kernels that expose events

- Wait to execute until two previous events complete.

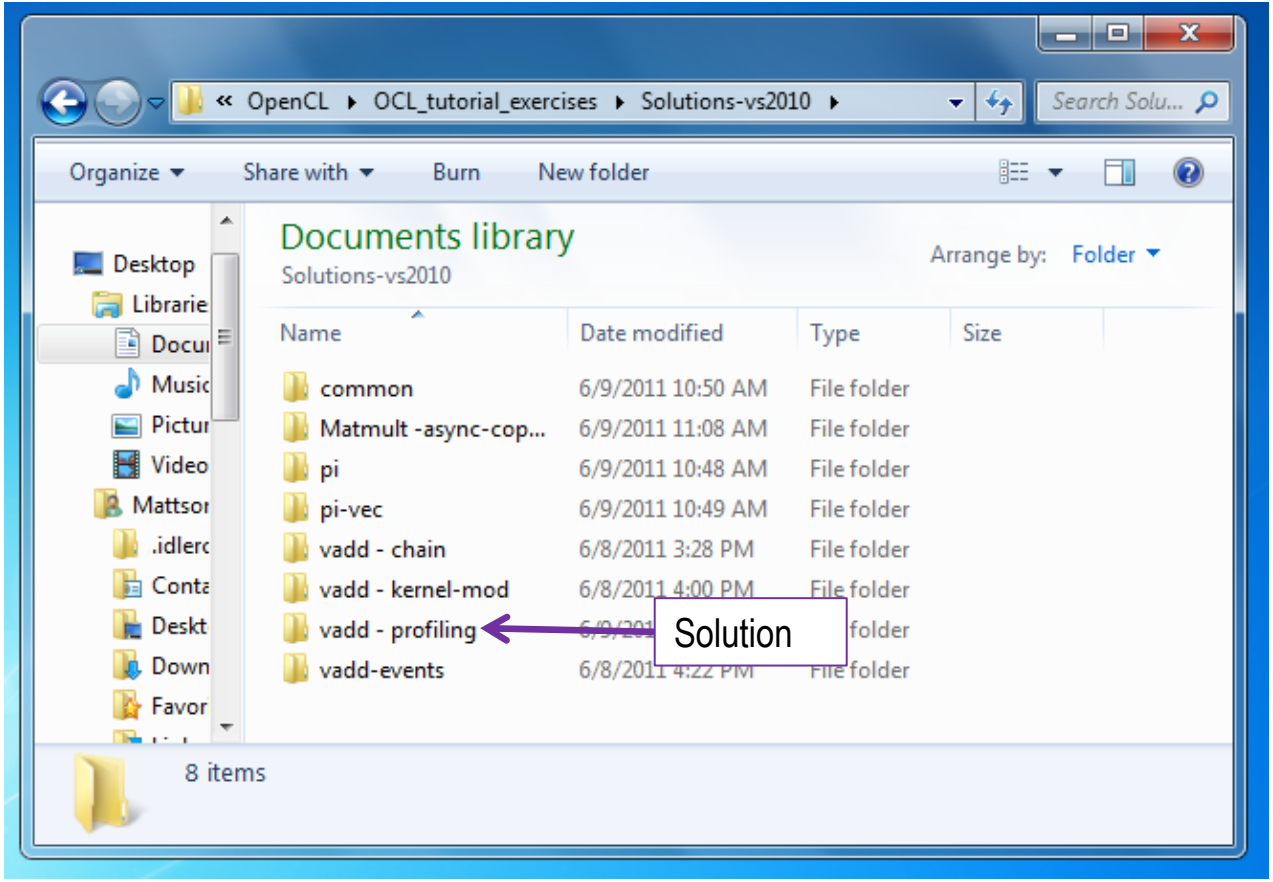
Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Events and Profiling OpenCL Commands

- **Work with your vadd with multiple kernels chained together**

- Create a queue with profiling enabled
- Use events to time the kernel execution.



Profiling with events

- **Create a command queue with profiling enabled**

```
commands = clCreateCommandQueue(context, device_id,
                                CL_QUEUE_PROFILING_ENABLE, &err)
```

- **Enqueue the command, but expose an event**

```
cl_event prof_event;
err = clEnqueueNDRangeKernel(commands, kernel, nd, NULL, global, NULL,
                             0, NULL, &prof_event);
```

- **Wait for the command to finish (using the event)**

```
err = clWaitForEvents( 1, &prof_event );
```

- **Extract timing data from the event**

```
cl_ulong ev_start_time=(cl_ulong)0;     cl_ulong ev_end_time=(cl_ulong)0;
err = clGetEventProfilingInfo(prof_event,  CL_PROFILING_COMMAND_START,
                              sizeof(cl_ulong),  &ev_start_time, NULL);
err = clGetEventProfilingInfo(prof_event,  CL_PROFILING_COMMAND_END,
                              sizeof(cl_ulong),   &ev_end_time,  NULL);
printf(“ runtime = %f secs “,(double) (ev_end_time – ev_start_time) *1.0e-9);
```

- **Other profiling info includes:**
  - CL_PROFILING_COMMAND_QUEUED, CL_PROFILING_COMMAND_SUBMIT

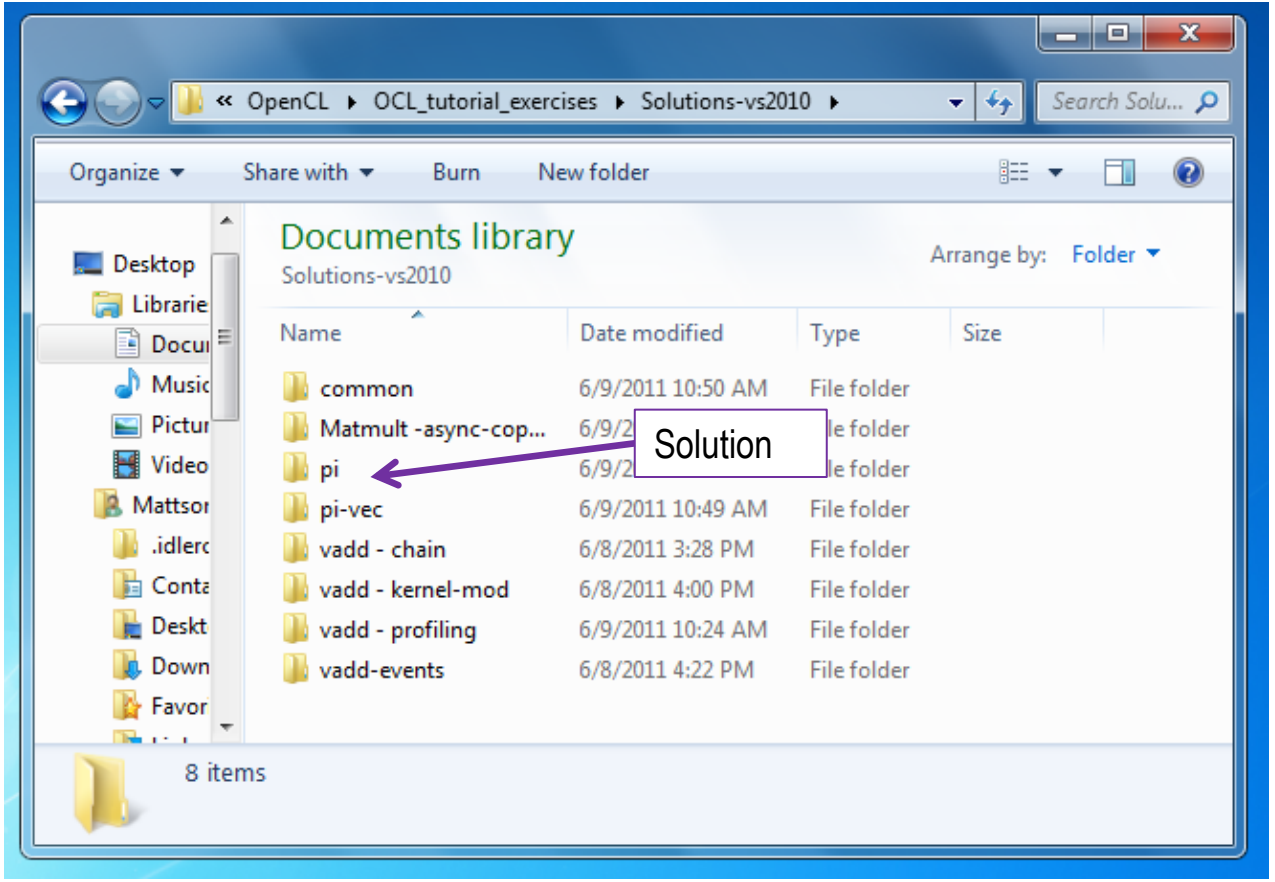
Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Writing your own program “from scratch”

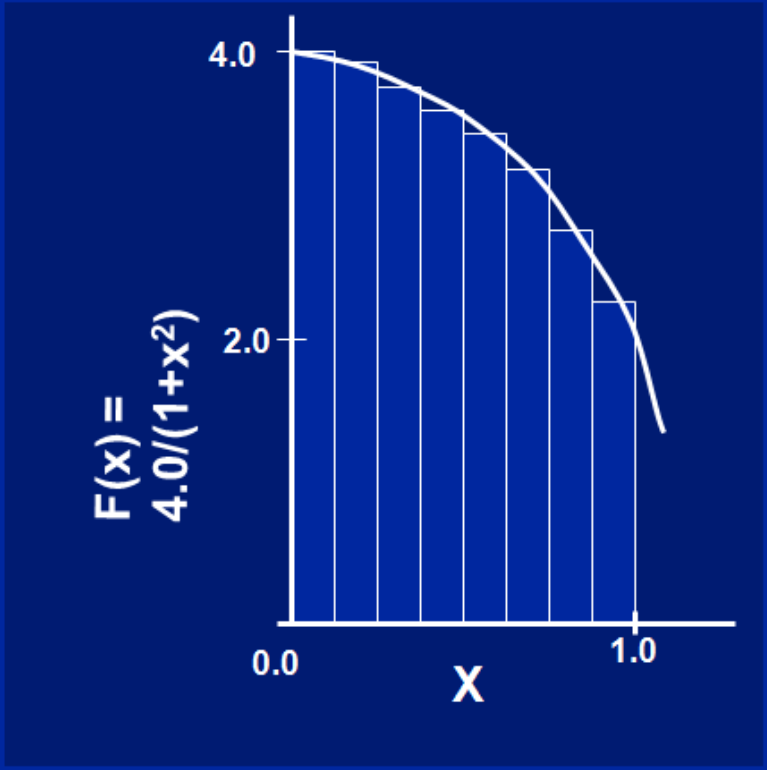
- **Start with the provided numerical integration program (pi)**

- Convert the pi program into an OpenCL kernel
- Create a host program to execute the kernel.



The pi program

Numerical Integration



Mathematically, we know that:

$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

We can approximate the integral as a sum of rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

The Starting point for this exercise

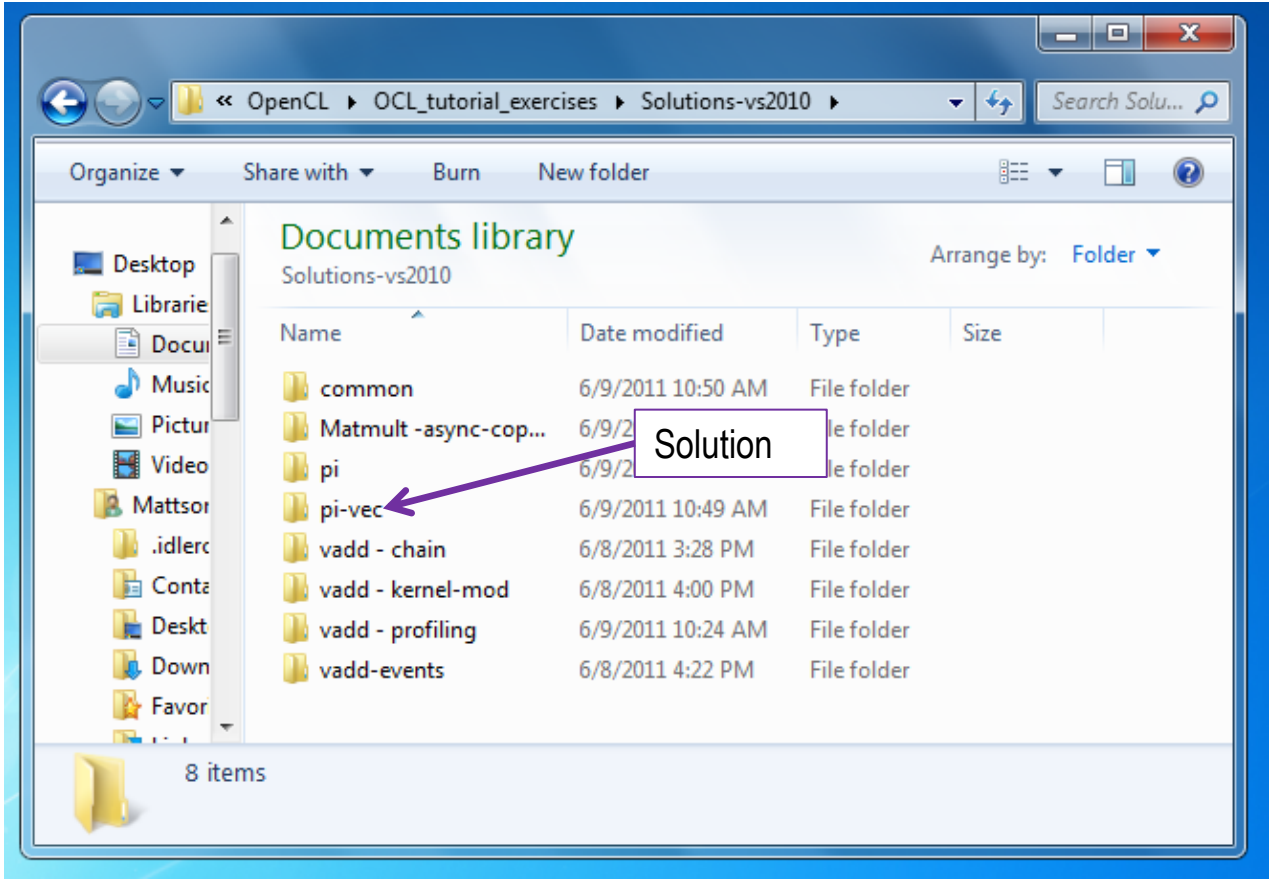
Serial PI Program

```
static long num_steps = 100000;
double step;
void main ()
{       int i;   double x, pi, sum = 0.0;

        step = 1.0/(double) num_steps;

        for (i=0;i< num_steps; i++){
                x = (i+0.5)*step;
                sum = sum + 4.0/(1.0+x*x);
        }
        pi = step * sum;
}
```

Working with vectors inside the kernel

- **Convert your "scalar" pi kernel program into one that uses vectors.**

- Hint: unroll loops to the width of the vectors.
- Convert inner loops to use OpenCL's vector instructions.



Exercises

- **Using your local OpenCL environment**
  - Run the provided vadd program
- **Working with queues**
  - Chain multiple vadds together
- **Modifying kernels**
  - Change vadd to add three vectors
- **Events and out of order queues**
  - force a partial order with vadd kernels using events
- **The OpenCL profiling interface**
  - Use events to profile commands
- **A full program on your own: local data and reductions.**
  - Pi program with Scalar kernels
  - Pi program with Vector kernels
- **Optimization of OpenCL programs**
  - Matrix multiplication … make it fast!

Optimizing OpenCL Programs

- **Start with the provided serial matrix multiplication program.**

- Convert the serial program into an OpenCL kernel
- Write a host program to run the kernel.\
- Optimize to make run as fast as you can on the platform of your choice.