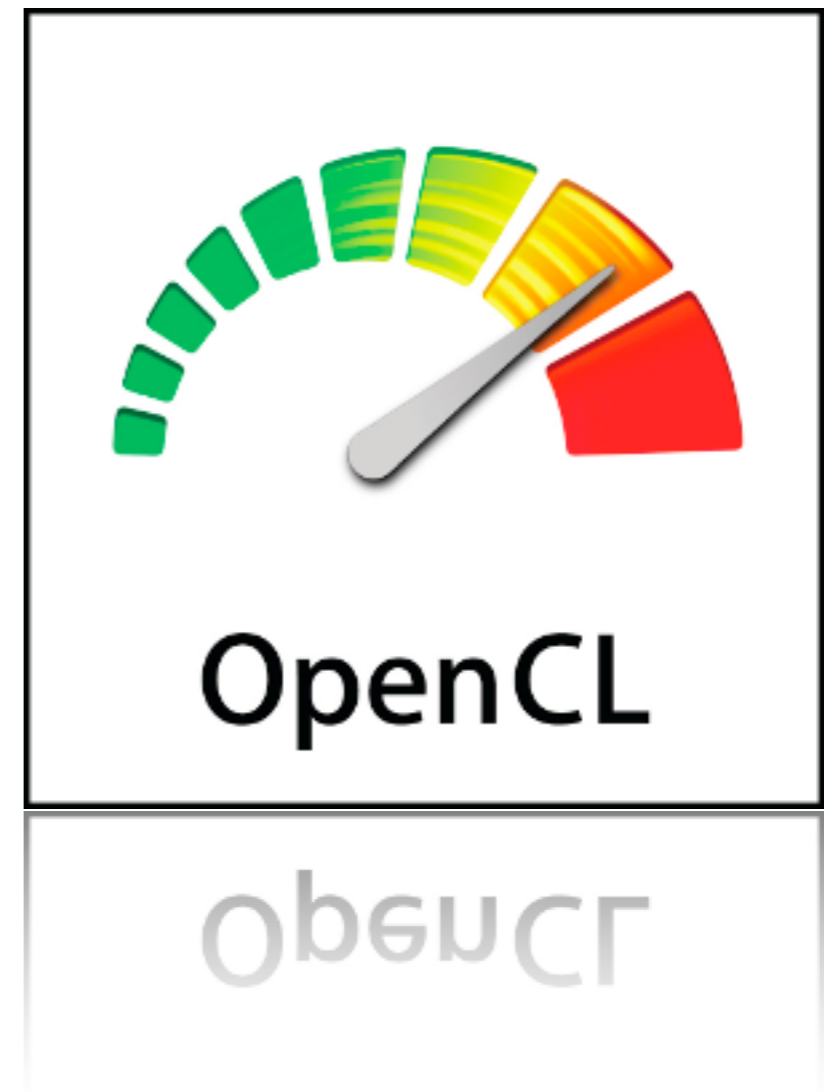


# Other Programming Models for GPUs

Fall 2012

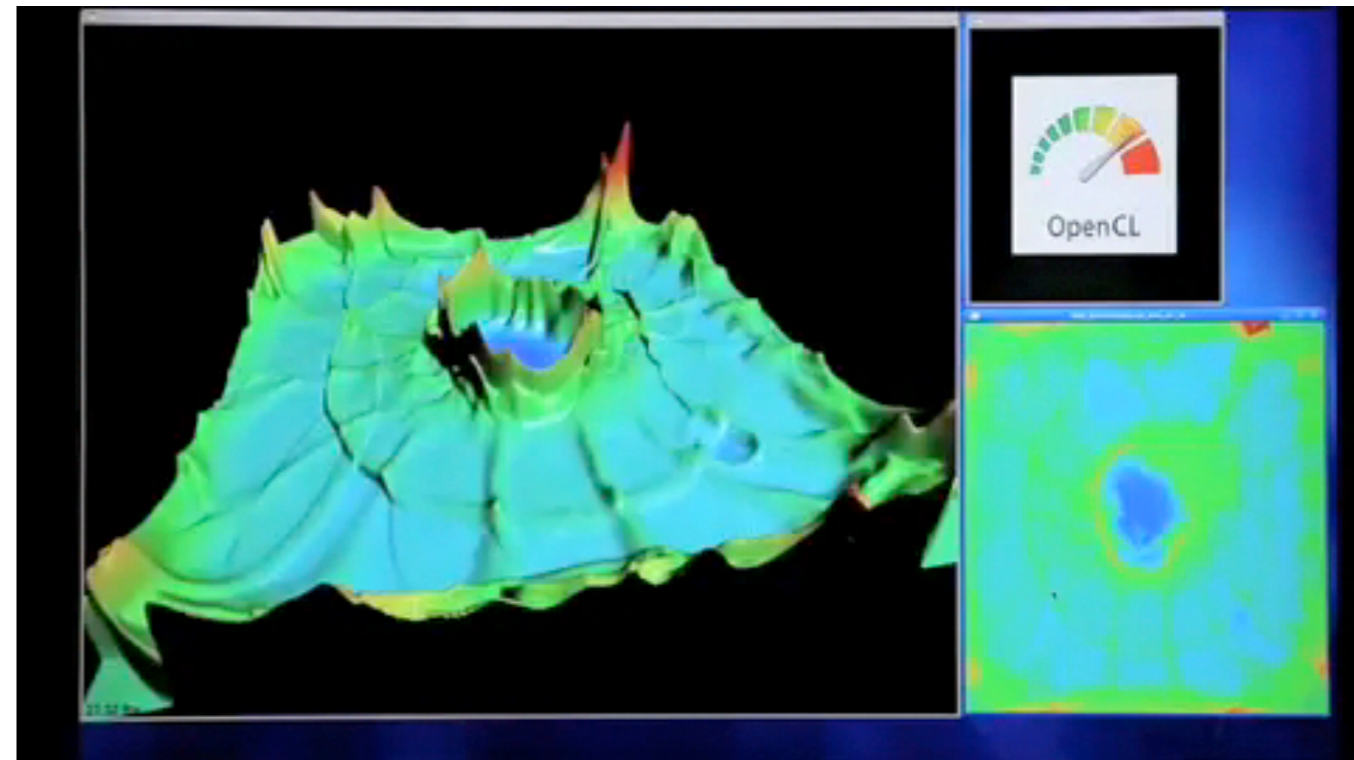
# OpenCL: Open standard

- Driven by Apple (comes standard in Snow Leopard)
- NVIDIA, AMD, Intel, IBM (Cell)
- Exposes a consistent, GPU-like interface to any multicore system



# Heterogeneous, Open

- Can work with various hardware
- IBM Cell, AMD processors, ATI cards, NVidia cards, Intel processors
- Multi- and Many- core
- SC09 demo: parallel CFD running on all of the above at once in *same program*, using MPI to tie them together



# All is not roses

- Code may be portable, but **performance** isn't
- (Autotuning?)
- For short-term future (~32 cores, cache-coherent), OpenMP is going to be the way to go for multicore CPUs



# All is not roses

- You pay for all that extra generality:



# Quick Reference Card

• Page 1

## OpenCL API 1.1 Quick Reference Card - Page 1

**OpenCL (Open Computing Language)** is a multi-vendor open standard for general-purpose parallel programming of heterogeneous systems that include CPUs, GPUs and other processors. OpenCL provides a uniform programming environment for software developers to write efficient, portable code for high-performance compute servers, desktop computer systems and handheld devices. [N.A.A] refers to the section in the API Specification available at [www.khronos.org/opencl](http://www.khronos.org/opencl).

### The OpenCL Runtime

**Command Queues [5.1]**  
`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  
`cl_int clRetainCommandQueue ( cl_command_queue command_queue)`  
`cl_int clReleaseCommandQueue ( cl_command_queue command_queue)`  
`cl_int clGetCommandQueueInfo ( cl_command_queue command_queue, cl_command_queue_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`  
param\_name: CL\_QUEUE\_CONTEXT, CL\_QUEUE\_DEVICE, CL\_QUEUE\_REFERENCE\_COUNT, CL\_QUEUE\_PROPERTIES

### Buffer Objects

Elements of a buffer object can be scalar or vector data type or a user-defined structure. Elements are stored sequentially and are accessed using a pointer by a kernel executing on a device. Data is stored in the same format as it is accessed by the kernel.

#### Create Buffer Objects [5.2.1]

`cl_mem clCreateBuffer ( cl_context context, cl_mem_flags flags, size_t size, void *host_ptr, cl_int *errcode_ret)`  
`cl_mem clCreateSubBuffer ( cl_mem buffer, cl_mem_flags flags, cl_buffer_create_type buffer_create_type, const void *buffer_create_info, cl_int *errcode_ret)`  
flags for `clCreateBuffer` and `clCreateSubBuffer`: CL\_MEM\_READ\_WRITE, CL\_MEM\_WRITE\_READ\_ONLY, CL\_MEM\_USE\_ALLOC\_COPYS\_HOST\_PTR

#### Read, Write, Copy Buffer Objects [5.2.2]

`cl_int clEnqueueReadBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_read, size_t offset, size_t cb, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`  
`cl_int clEnqueueWriteBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_write, size_t offset, size_t cb, const void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

### Program Objects

#### Create Program Objects [5.4.1]

`cl_program clCreateProgramWithSource ( cl_context context, cl_uint count, const char **strings, const size_t *lengths, cl_int *errcode_ret)`  
`cl_program clCreateProgramWithBinary ( cl_context context, cl_uint num_devices, const cl_device_id *device_list, const unsigned char **binaries, cl_int *binary_status, cl_int *errcode_ret)`  
`cl_int clRetainProgram ( cl_program program)`  
`cl_int clReleaseProgram ( cl_program program)`

### The OpenCL Platform Layer

The OpenCL platform layer implements platform-specific features that allow applications to query OpenCL devices, device configuration information, and to create OpenCL contexts using one or more devices.

#### Contexts [4.1]

`cl_context clCreateContext ( const cl_context_properties *properties, cl_uint num_devices, const cl_device_id *devices, void (CL_CALLBACK *pfn_notify) (const char *errmsg, const void *private_info, size_t cb, void *user_data), void *user_data, cl_int *errcode_ret)`  
properties: CL\_CONTEXT\_PLATFORM, CL\_GL\_CONTEXT\_KHR, CL\_CGL\_SHAREGROUP\_KHR, CL\_D3D10\_DISPLAY\_KHR, CL\_WGL\_HDC\_KHR  
`cl_context clCreateContextFromType ( const cl_context_properties *properties, cl_device_type device_type, void (CL_CALLBACK *pfn_notify) (const char *errmsg, const void *private_info, size_t cb, void *user_data), void *user_data, cl_int *errcode_ret)`  
properties: See `clCreateContext`  
`cl_int clRetainContext ( cl_context context)`  
`cl_int clReleaseContext ( cl_context context)`  
`cl_int clGetContextInfo ( cl_context context, cl_context_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`  
param\_name: CL\_CONTEXT\_REFERENCE\_COUNT, CL\_CONTEXT\_DEVICES, CL\_CONTEXT\_NUM\_DEVICES

#### Querying Platform Info and Devices [4.1, 4.2]

`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, CL\_DEVICE\_MAX\_WORK\_GROUP\_SIZE, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_CHAR, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_INT, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_LONG, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_FLOAT, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_DOUBLE, CL\_DEVICE\_NATIVE\_PREFERRED\_VECTOR\_WIDTH\_HALF, CL\_DEVICE\_MAX\_CLOCK\_FREQUENCY, CL\_DEVICE\_ADDRESS\_BITS, CL\_DEVICE\_MAX\_MEM\_ALLOC\_SIZE, CL\_DEVICE\_IMAGE\_SUPPORT, CL\_DEVICE\_MAX\_READ\_WRITE\_IMAGE\_ARGS, CL\_DEVICE\_IMAGE2D\_MAX\_WIDTH\_HEIGHT, CL\_DEVICE\_IMAGE3D\_MAX\_WIDTH\_HEIGHT\_DEPTH, CL\_DEVICE\_MAX\_SAMPLERS, CL\_DEVICE\_MAX\_PARAMETER\_SIZE, CL\_DEVICE\_MEM\_BASE\_ADDR\_ALIGN, CL\_DEVICE\_MIN\_DATA\_TYPE\_ALIGN\_SIZE, CL\_DEVICE\_SINGLE\_FP\_CONFIG, CL\_DEVICE\_GLOBAL\_MEM\_CACHE\_TYPE\_SIZE, CL\_DEVICE\_GLOBAL\_MEM\_CACHELINE\_SIZE, CL\_DEVICE\_GLOBAL\_MEM\_SIZE, CL\_DEVICE\_MAX\_CONSTANT\_BUFFER\_SIZE\_ARGS, CL\_DEVICE\_LOCAL\_MEM\_TYPE\_SIZE, CL\_DEVICE\_ERROR\_CORRECTION\_SUPPORT, CL\_DEVICE\_PROFILING\_TIMER\_RESOLUTION, CL\_DEVICE\_ENDIAN\_LITTLE, CL\_DEVICE\_AVAILABLE, CL\_DEVICE\_COMPILER\_AVAILABLE, CL\_DEVICE\_EXECUTION\_CAPABILITIES, CL\_DEVICE\_QUEUE\_PROPERTIES, CL\_DEVICE\_NAME, VENDOR, PROFILE, EXTENSIONS, CL\_DEVICE\_HOST\_UNIFIED\_MEMORY, CL\_DEVICE\_OPENCL\_C\_VERSION, CL\_DEVICE\_VERSION, CL\_DRIVER\_VERSION, CL\_DEVICE\_PLATFORM

### Map Buffer Objects [5.2.2]

`void * clEnqueueMapBuffer ( cl_command_queue command_queue, cl_mem buffer, cl_bool blocking_map, cl_map_flags map_flags, size_t offset, size_t cb, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)`

#### Map Buffer Objects [5.4.1-2]

`cl_int clRetainMemObject ( cl_mem memobj)`  
`cl_int clReleaseMemObject ( cl_mem memobj)`  
`cl_int clSetMemObjectDestructorCallback ( cl_mem memobj, void (CL_CALLBACK *pfn_notify) ( cl_mem memobj, void *user_data), void *user_data)`  
`cl_int clEnqueueUnmapMemObject ( cl_command_queue command_queue, cl_mem memobj, void *mapped_ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

#### Query Buffer Object [5.4.3]

`cl_int clGetMemObjectInfo ( cl_mem memobj, cl_mem_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`  
param\_name: CL\_MEM\_TYPE, FLAGS, SIZE, HOST\_PTR, CL\_MEM\_MAP\_REFERENCE\_COUNT, CL\_MEM\_OFFSET, CL\_MEM\_CONTEXT, CL\_MEM\_ASSOCIATED\_MEMOBJECT

**Math Intrinsic:**  
-cl-single-precision-constant -cl-constants-are-zero  
**Warning request/suppress:**  
-w -Werror  
**Control OpenCL C language version:**  
-cl-std=CL1.1 // OpenCL 1.1 specification.

**Build Program Executable [5.4.2]**  
`cl_int clBuildProgram ( cl_program program, cl_uint num_devices, const cl_device_id *device_list, const char *options, void (CL_CALLBACK *pfn_notify) ( cl_program program, void *user_data), void *user_data)`  
**Build Options [5.4.3]**  
-D name -D name=definition -d dir  
**Optimization options:**  
-cl-opt-disable -cl-strict-aliasing  
-cl-mad-enable -cl-no-signed-zeros  
-cl-finite-math-only -cl-fast-relaxed-math  
-cl-unroll-math-optimizations

**Query Program Objects [5.4.4]**  
`cl_int clGetProgramInfo ( cl_program program, cl_program_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`  
param\_name: CL\_PROGRAM\_REFERENCE\_COUNT, CL\_PROGRAM\_CONTEXT, NUM\_DEVICES, DEVICES, CL\_PROGRAM\_SOURCE, BINARY\_SIZES, BINARIES

(Program Objects Continue >)

# Quick Reference Card

• Page 2

## OpenCL API 1.1 Quick Reference Card - Page 2

### Program Objects (continued)

```

d_int clGetProgramBuildInfo (cl_program program,
    cl_device_id device, cl_program_build_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
param_name: CL_PROGRAM_BUILD_STATUS, OPTIONS, LOG)
Unload the OpenCL Compiler (3.6.4)
d_int clUnloadCompiler (void)
    
```

### Supported Data Types

#### Built-in Scalar Data Types (3.1.1)

OpenCL Type	API Type	Description
bool	--	true (1) or false (0)
char	cl_char	8-bit signed
unsigned char, uchar	cl_uchar	8-bit unsigned
short	cl_short	16-bit signed
unsigned short, ushort	cl_ushort	16-bit unsigned
int	cl_int	32-bit signed
unsigned int, uint	cl_uint	32-bit unsigned
long	cl_long	64-bit signed
unsigned long, ulong	cl_ulong	64-bit unsigned
float	cl_float	32-bit float
half	cl_half	16-bit float (for storage only)
size_t	--	32- or 64-bit unsigned integer
ptrdiff_t	--	32- or 64-bit signed integer
intptr_t	--	signed integer
uintptr_t	--	unsigned integer
void	void	void

#### Built-in Vector Data Types (3.1.2)

OpenCL Type	API Type	Description
charn	cl_charn	8-bit signed
ucharn	cl_ucharn	8-bit unsigned
shortn	cl_shortn	16-bit signed
ushortn	cl_ushortn	16-bit unsigned
intn	cl_intn	32-bit signed
uintn	cl_uintn	32-bit unsigned
longn	cl_longn	64-bit signed
ulongn	cl_ulongn	64-bit unsigned
floatn	cl_floatn	32-bit float

#### Other Built-in Data Types (3.1.3)

OpenCL Type	Description
image2d_t	2D image handle
image3d_t	3D image handle
sampler_t	sampler handle
event_t	event handle

#### Reserved Data Types (3.1.4)

OpenCL Type	Description
booln	boolean vector
double, doublen	OPTIONAL 64-bit float, vector
halfn	16-bit, vector
quadd, quaddn	128-bit float, vector
complex_half, complex_halfn	imaginary half, imaginary halfn
complex_float, complex_floatn	imaginary float, imaginary floatn
complex_double, complex_double	imaginary double, imaginary double
complex_quad, complex_quadn	imaginary quad, imaginary quadn
floatnmx	n*m matrix of 32-bit floats
doublenmx	n*m matrix of 64-bit floats
long_double, long_double	64-128-bit float, vector
long_long, long_longn	128-bit signed
unsigned_long_long, unsigned_long_long	128-bit unsigned

### Kernel and Event Objects

```

Create Kernel Objects (3.7.1)
cl_kernel clCreateKernel (cl_program program,
    const char *kernel_name, d_int *errcode_ret)
cl_int clCreateKernelsInProgram (cl_program program,
    d_uint num_kernels, cl_kernel *kernels,
    d_uint *num_kernels_ret)
cl_int clRetainKernel (cl_kernel kernel)
cl_int clReleaseKernel (cl_kernel kernel)
    
```

#### Kernel Args. & Object Queries (3.7.2, 3.7.3)

```

cl_int clSetKernelArg (cl_kernel kernel, d_uint arg_index,
    size_t arg_size, const void *arg_value)
cl_int clGetKernelInfo (cl_kernel kernel,
    cl_kernel_info param_name, size_t param_value_size,
    void *param_value, size_t *param_value_size_ret)
param_name: CL_KERNEL_FUNCTION_NAME,
    CL_KERNEL_NUM_ARGS, CL_KERNEL_REFERENCE_COUNT,
    CL_KERNEL_CONTEXT, CL_KERNEL_PROGRAM
cl_int clGetKernelWorkGroupInfo (
    cl_kernel kernel, cl_device_id device,
    d_kernel_work_group_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
param_name: CL_KERNEL_WORK_GROUP_SIZE,
    CL_KERNEL_COMPILE_WORK_GROUP_SIZE,
    CL_KERNEL_LOCAL_PRIVATE_MEM_SIZE,
    CL_KERNEL_PREFERRED_WORK_GROUP_SIZE_MULTIPLE
    
```

#### Execute Kernels (3.8)

```

cl_int clEnqueueNDRangeKernel (
    cl_command_queue command_queue,
    cl_kernel kernel, d_uint work_dim,
    const size_t *global_work_offset,
    const size_t *global_work_size,
    const size_t *local_work_size,
    d_uint num_events_in_wait_list,
    const cl_event *event_wait_list, d_event *event)
cl_int clEnqueueTask (
    cl_command_queue command_queue, cl_kernel
    kernel, d_uint num_events_in_wait_list,
    const cl_event *event_wait_list, d_event *event)
cl_int clEnqueueNativeKernel (cl_command_queue
    command_queue, void (*user_func)(void *),
    void *args, size_t cb_args, d_uint num_mem_objects,
    const cl_mem *mem_list, const void **arg_mem_loc,
    d_uint num_events_in_wait_list,
    const cl_event *event_wait_list, d_event *event)
    
```

### Event Objects (3.9)

```

cl_event clCreateUserEvent (cl_context context,
    cl_int *errcode_ret)
cl_int clSetUserEventStatus (cl_event event,
    cl_int execution_status)
cl_int clWaitForEvents (cl_uint num_events,
    const cl_event *event_list)
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)
param_name: CL_EVENT_COMMAND_QUEUE, TYPE,
    CL_EVENT_CONTEXT, REFERENCE_COUNT,
    CL_EVENT_COMMAND_EXECUTION_STATUS
cl_int clSetEventCallback (cl_event event,
    cl_int command_exec_callback_type,
    void (CL_CALLBACK *fn)(cl_event, void *)
    (cl_event event, d_int event_command_exec_status,
    void *user_data),
    void *user_data)
cl_int clRetainEvent (cl_event event)
cl_int clReleaseEvent (cl_event event)
    
```

### Out-of-order Execution of Kernels & Memory Object Commands (3.10)

```

cl_int clEnqueueMarker (
    cl_command_queue command_queue,
    cl_event *event)
cl_int clEnqueueWaitForEvents (
    cl_command_queue command_queue,
    cl_uint num_events, const cl_event *event_list)
cl_int clEnqueueBarrier (
    cl_command_queue command_queue)
    
```

### Profiling Operations (3.11)

```

cl_int clGetEventProfilingInfo (cl_event event,
    cl_profiling_info param_name,
    size_t param_value_size, void *param_value,
    size_t *param_value_size_ret)
param_name: CL_PROFILING_COMMAND_QUEUED,
    CL_PROFILING_COMMAND_SUBMIT, START, END)
    
```

### Flush and Finish (3.12)

```

cl_int clFlush (cl_command_queue command_queue)
cl_int clFinish (cl_command_queue command_queue)
    
```

### Vector Component Addressing (3.1.7)

#### Vector Components

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
float2 v;	vx, xs0	vy, ys1														
float3 v;	vx, xs0	vy, ys1	vz, vs2													
float4 v;	vx, xs0	vy, ys1	vz, vs2	vw, vs3												
float8 v;	xs0	xs1	xs2	xs3	xs4	xs5	xs6	xs7								
float16 v;	xs0	xs1	xs2	xs3	xs4	xs5	xs6	xs7	xs8	xs9	xs10	xs11	xs12	xs13	xs14	xs15

#### Vector Addressing Equivalencies

Numeric indices are preceded by the letter s or i, e.g.: s1. Swizzling, duplication, and nesting are allowed, e.g.: vxv, vxv, v10.x

	v10	v11	vodd	v-even	v10	v11	vodd	v-even
float2	vx, xs0	vy, ys1	vx, xs1	vy, ys0	xs0123	xs4567	xs1357	xs2468
float3	xs01, vxv	xs23, vsw	xs13, vsyw	xs02, vsxz	xs01234567	xs89abcd	xs135789ab	xs2468facc
float6	xs01, vxv	xs23, vsw	xs13, vsyw	xs02, vsxz				

\*When using i or j with a 3-component vector, the k component is undefined.

#### Conversions & Type Casting Examples (3.2)

```

T a = (T)b; // Scalar to scalar, or scalar to vector
T a = convert_T(b);
T a = convert_T_R(b);
T a = as_T(b);
T a = convert_T_sat_R(b); //R is rounding mode
    
```

It can be one of the following rounding modes:  
 \_rtz toward zero  
 \_rtn toward -infinity  
 \_rtu toward +infinity

#### Operators (3.3)

These operators behave similarly as in C99 except that operands may include vector types when possible:

```

+ - * % / -- ++ == != &
~ ^ > < >= <= | ! && ||
?: >> << , = op= sizeof
    
```

#### Address Space Qualifiers (3.5)

```

__global, global __local, local
__constant, constant __private, private
    
```

#### Function Qualifiers (3.7)

```

__kernel, kernel
__attribute__((vec_type_hint(type))) //type defaults to int
__attribute__((work_group_size_hint(X, Y, Z)))
__attribute__((req_work_group_size(X, Y, Z)))
    
```

# Quick Reference Card

• Page 3

## OpenCL API 1.1 Quick Reference Card - Page 3

Each occurrence of T within a function call must be the same. n is 2, 3, 4, 8, or 16 unless otherwise specified.

### Preprocessor Directives & Macros (9.10)

#pragma OPENCL\_FP\_CONTRACT on/off-switch  
on/off-switch: ON, OFF, DEFAULT

__FILE__	Current source file
__LINE__	Integer line number
__OPENCL_VERSION__	Integer version number
__CL_VERSION_1_0__	Substitutes integer 100 for version 1.0
__CL_VERSION_1_1__	Substitutes integer 110 for version 1.1
__ENDIAN_LITTLE__	1 if device is little endian
__kernel_exec(X, type)	Same as: kernel_attribute { (work_group_size_hint(X, 1, 1))   attribute { (vec_type_hint)(type) }
__IMAGE_SUPPORT__	1 if images are supported
__FAST_RELAXED_MATH__	1 if -cl-fast-relaxed-math optimization option is specified

### Specify Type Attributes (9.10.1)

Use to specify special attributes of enum, struct and union types.

__attribute__((aligned(n)))	__attribute__((endian(host)))
__attribute__((aligned))	__attribute__((endian(device)))
__attribute__((packed))	__attribute__((endian))

### Math Constants (9.11.2)

The values of the following symbolic constants are type float and are accurate within the precision of a single precision floating point number.

HUGE_VAL	Positive double expression, evals to +infinity. Used as error value. <b>obsolete</b>	M_LN2_F	Value of log <sub>2</sub>
INFINITY	Constant float expression, positive or unsigned infinity	M_LN10_F	Value of log <sub>10</sub>
MAXFLOAT	Value of max. non-infinity single-precision floating-point number	M_PI_F	Value of π
HUGE_VALF	Positive float expression, evaluates to +infinity. Used as error value	M_PI_2_F	Value of π / 2
M_E_F	Value of e	M_PI_4_F	Value of π / 4
M_LOG2E_F	Value of log <sub>2</sub> e	M_1_PI_F	Value of 1 / π
M_LOG10E_F	Value of log <sub>10</sub> e	M_2_PI_F	Value of 2 / π
		M_2_SQRTPI_F	Value of 2 / √π
		M_SQRT2_F	Value of √2
		M_SQRT2_2_F	Value of 1 / √2

### Work-Item Built-in Functions (9.11.1) D is dimension index.

uint get_work_dim()	Num. of dimensions in use	size_t get_local_id(int D)	Local work-item ID
size_t get_global_size(int D)	Num. of global work-items	size_t get_num_groups(int D)	Num. of work-groups
size_t get_group_id(int D)	Global work-item ID value	size_t get_group_id(int D)	Returns the work-group ID
size_t get_local_size(int D)	Num. of local work-items	size_t get_global_offset(int D)	Returns global offset

### Integer Built-in Functions (9.11.3)

T is type char, charn, uchar, ushort, short, shorts, ushort, int, ints, uint, uints, long, longs, ulong, or ulongts. S is the unsigned version of T. S is the scalar version of Z.

Uabs(T x)	x
Uabs_diff(T x, T y)	x - y  without module overflow
Uadd_sat(T x, T y)	x + y and saturates the result
Uadd(T x, T y)	(x + y) >> 1 without mod overflow
Uadd(T x, T y)	(x + y + 1) >> 1
Uadd(T x, T y)	(x + y + 1) >> 1
Udla(T x)	Number of leading 0 bits in x
Uclamp(T x, T min, T max)	min(max(x, minval), maxval)
Uclamp(T x, S min, S max)	min(max(x, minval), maxval)
Umod_n(T a, T b, T c)	mul_n(a, b) + c
Umod_sat(T a, T b, T c)	a * b + c and saturates the result
Umax(T x, T y)	y if x < y, otherwise z returns x
Umax(T x, S y)	y if x < y, otherwise z returns x
Umin(T x, T y)	y if y < x, otherwise z returns x
Umin(T x, S y)	y if y < x, otherwise z returns x
Umul_n(T x, T y)	high half of the product of x and y
Urotate(T x, T d)	result[ndb + x] [ndb] or (ndb)

### T ash\_sat(T x, T y)

x - y and saturates the result

For example, scalar types are permitted for the vector types below.

shortn_upsample (shortn h, ushortn k)	result[i] = (shortn)h[(i+k)/2][i]
ushortn_upsample (ushortn h, ushortn k)	result[i] = (ushortn)h[(i+k)/2][i]
intn_upsample (intn h, uintn k)	result[i] = (intn)h[(i+k)/2][i]
uintn_upsample (uintn h, uintn k)	result[i] = (uintn)h[(i+k)/2][i]
longn_upsample (longn h, uintn k)	result[i] = (longn)h[(i+k)/2][i]
ulongn_upsample (ulongn h, uintn k)	result[i] = (ulongn)h[(i+k)/2][i]

The following fast integer functions optimize the performance of kernels. In these functions, T is type int, int2, int3, int4, int8, int16, uint, uint2, uint4, uint8, or uint16.

Umod24(T a, T b, T c)	Multiply 24-bit int. values a, b, add 32-bit int. result to 32-bit int. c
Umul24(T a, T b)	Multiply 24-bit int. values a and b

### Common Built-in Functions (9.11.4)

T is type float or floatn (or optionally double, doublen, or halfn). Optional extensions enable double, doublen, and halfn types.

T clamp(T x, T min, T max)	Clamp x to range given by min, max
floatn_clamp(floatn x, float min, float max)	Clamp x to range given by min, max
doublen_clamp(doublen x, double min, double max)	Clamp x to range given by min, max
halfn_clamp(halfn x, half min, half max)	Clamp x to range given by min, max
T degrees(T radians)	radians to degrees
T max(T x, T y)	Max of x and y
floatn_max(floatn x, float y)	Max of x and y
doublen_max(doublen x, double y)	Max of x and y
halfn_max(halfn x, half y)	Max of x and y
T min(T x, T y)	Min of x and y
floatn_min(floatn x, float y)	Min of x and y
doublen_min(doublen x, double y)	Min of x and y
halfn_min(halfn x, half y)	Min of x and y
T mix(T x, T y, T a)	Linear blend of x and y
floatn_mix(floatn x, float y, float a)	Linear blend of x and y
doublen_mix(doublen x, double y, double a)	Linear blend of x and y
halfn_mix(halfn x, half y, half a)	Linear blend of x and y
T radians(T degrees)	degrees to radians
T step(T edge, T a)	0.0 if x < edge, else 1.0
floatn_step(float edge, floatn x)	0.0 if x < edge, else 1.0
doublen_step(double edge, doublen x)	0.0 if x < edge, else 1.0
halfn_step(half edge, halfn x)	0.0 if x < edge, else 1.0
T smoothstep(T edge0, T edge1, T x)	Step and interpolate
floatn_smoothstep(float edge0, float edge1, floatn x)	Step and interpolate
doublen_smoothstep(double edge0, double edge1, doublen x)	Step and interpolate
halfn_smoothstep(half edge0, half edge1, halfn x)	Step and interpolate
T sign(T x)	Sign of x

### Math Built-in Functions (9.11.2)

T is type float or floatn (or optionally double, doublen, or halfn). intn, uintn, and ulongn must be scalar when T is scalar. Q is qualifier \_\_global\_\_, \_\_local\_\_, or \_\_private. HN indicates that half and native variants are available by prepending "half\_" or "native\_" to function names. Prototypes shown in purple are half\_ and native\_ only. Optional extensions enable double, doublen, half, and halfn types.

T acos(T)	Arc cosine	T expm1(T x)	e <sup>x</sup> - 1.0	T minmag(T x, T y)	Minimum magnitude of x and y
T acosh(T)	Inverse hyperbolic cosine	T fabs(T)	Absolute value	T modf(T x, Q T *int)	Decompose a floating-point number
T acosh(T x)	acos(x) / π	T fdim(T x, T y)	"Positive difference" between x and y	float nan (uint noncode)	Quiet NaN
T asin(T)	Arc sine	T floor(T)	Round to integer toward -infinity	float nan (uint noncode)	Quiet NaN
T asinh(T)	Inverse hyperbolic sine	T fma(T a, T b, T c)	Multiply and add, then round	float nan (ulong noncode)	Quiet NaN
T asinh(T x)	asin(x) / π	T fmax(T x, T y)	Return y if x < y, otherwise it returns x	double nan (ulong noncode)	Quiet NaN
T atan(T y, over_s)	Arc tangent	halfn_fmax(halfn x, halfn y)	Return y if y < x, otherwise it returns x	T nextafter(T x, T y)	Next representable floating-point value following x in the direction of y
T atan2(T y, T x)	Arc tangent of y/x	floatn_fmax(floatn x, floatn y)	Return y if y < x, otherwise it returns x	T pow(T x, T y)	Compute x to the power of y (x <sup>y</sup> )
T atanh(T)	Hyperbolic arc tangent	doublen_fmax(doublen x, doublen y)	Return y if y < x, otherwise it returns x	T powi(T x, intn y)	Compute x <sup>y</sup> , where y is an integer
T atan2pi(T x, T y)	atan2(x, y) / π	T fmin(T x, T y)	Return y if y < x, otherwise it returns x	T power(T x, T y)	HN Compute x <sup>y</sup> , where x is ≠ 0
T cbrt(T)	Cube root	halfn_fmin(halfn x, halfn y)	Return y if y < x, otherwise it returns x	T half_recip(T x)	1 / x [T may be float or floatn]
T ceil(T)	Round to integer toward +infinity	floatn_fmin(floatn x, floatn y)	Return y if y < x, otherwise it returns x	T native_recip(T x)	1 / x [T may be float or floatn]
T copy_sign(T x, T y)	x with sign changed to sign of y	doublen_fmin(doublen x, doublen y)	Return y if y < x, otherwise it returns x	T remainder(T x, T y)	Floating point remainder
T cos(T)	HN Cosine	T fmod(T x, T y)	Modulus. Returns x - y * trunc(x/y)	T remquo(T x, T y, Q intn *quo)	Floating point remainder and quotient
T cosh(T)	Hyperbolic cosine	T fract(T x, Q T *int)	Fractional value in x	T rint(T)	Round to nearest, even integer
T cospi(T x)	cos(π x)	T fexp(T x, Q intn *exp)	Extract mantissa and exponent	T roots(T x, intn y)	Compute x to the power of 1/y
T half_divide(T x, T y)	x / y [T may be float or floatn]	T hypot(T x, T y)	Square root of x <sup>2</sup> + y <sup>2</sup>	T round(T x)	Integral value nearest to x rounding
T native_divide(T x, T y)	x / y [T may be float or floatn]	intn_logb(T x)	Return exponent as an integer value	T rsqrt(T)	HN Inverse square root
T erf(T)	Complementary error function	T ldexp(T x, intn n)	x * 2 <sup>n</sup>	T sin(T)	HN Sine
T erf(T)	Calculates error function of T	T ldexp(T x, int n)	x * 2 <sup>n</sup>	T sinh(T)	HN Hyperbolic sine
T exp(T x)	HN Exponential base e	T lgamma(T x)	Log gamma function	T sinhpi(T x, Q T *cospi)	Sine and cosine of x
T exp2(T)	HN Exponential base 2	T gamma_r(T x, Q intn *rpo)	Log gamma function	T sinh(T)	Hyperbolic sine
T exp10(T)	HN Exponential base 10	T log(T)	HN Natural logarithm	T sinpi(T x)	sin(π x)
		T log2(T)	HN Base 2 logarithm	T sqrt(T)	HN Square root
		T log10(T)	HN Base 10 logarithm	T tan(T)	HN Tangent
		T log2p(T x)	ln(2.0 * x)	T tanh(T)	Hyperbolic tangent
		T logb(T x)	Exponent of x	T tanhpi(T x)	tan(π x)
		T mad(T a, T b, T c)	Approximates a * b + c	T tgamma(T)	Gamma function
		T maxmag(T x, T y)	Maximum magnitude of x and y	T trunc(T)	Round to integer toward zero





# Quick Reference Card

• Page 4

## OpenCL API 1.1 Quick Reference Card - Page 4

Geometric Built-in Functions [6.11.5]		Vector distance		Normal vector length 1		
<p>Vector types may have 2, 3, or 4 components. Optional extensions enable double, double_n, and half_n types.</p>						
float dot (float p0, float p1)	Dot product	float distance (float p0, float p1)	Vector distance	float normalize (float p)	Normal vector length 1	
float dot (floatn p0, floatn p1)		float distance (floatn p0, floatn p1)		floatn normalize (floatn p)		
double dot (double p0, double p1)		double distance (double p0, double p1)		double normalize (double p)		
double dot (doublen p0, doublen p1)		double distance (doublen p0, doublen p1)		half normalize (half p)		
half dot (half p0, half p1)		half distance (half p0, half p1)		half normalize (halfn p)		
float dot (halfn p0, halfn p1)	Cross product	float length (float p)	Vector length	float fast_distance (float p0, float p1)	Vector distance	
double dot (halfn p0, halfn p1)		float length (floatn p)		float fast_distance (floatn p0, floatn p1)		
float(3,4) cross (float(3,4) p0, float(3,4) p1)		double length (double p)		float fast_length (float p)		Vector length
double(3,4) cross (double(3,4) p0, double(3,4) p1)		double length (doublen p)		float fast_length (floatn p)		
half(3,4) cross (half(3,4) p0, half(3,4) p1)		half length (half p)		float fast_normalize (float p)		
	half length (halfn p)	floatn fast_normalize (floatn p)				

Relational Built-in Functions [6.11.6]		Test for +ve or -ve infinity	
<p>T is type float, floatn, char, charn, uchar, uchar_n, short, shorts, ushort, ushortn, int, intn, sint, sintn, long, longn, ulong, ulongn (and optionally double, double_n). S is type char, charn, short, shorts, int, intn, long, or longn. Q is type uchar, uchar_n, ushort, ushortn, uint, uintn, ulong, or ulongn. Optional extensions enable double, double_n, and half_n types.</p>			
int isequal (float x, float y)	Compare of x == y	int isnan (float)	Test for a NaN
intn isequal (floatn x, floatn y)		intn isnan (floatn)	
double isequal (double x, double y)		int isnan (double)	
long isequal (doublen x, doublen y)		long isnan (doublen)	
int isequal (half x, half y)		int isnan (half)	
short isequal (halfn x, halfn y)	Compare of x != y	int isnormal (float)	Test for a normal value
intn isequal (floatn x, floatn y)		intn isnormal (floatn)	
double isequal (double x, double y)		int isnormal (double)	
long isequal (doublen x, doublen y)		long isnormal (doublen)	
int isequal (half x, half y)		int isnormal (half)	
short isequal (halfn x, halfn y)	Compare of x > y	int isordered (float x, float y)	Test if arguments are ordered
intn isgreater (floatn x, floatn y)		intn isordered (floatn x, floatn y)	
double isgreater (double x, double y)		double isordered (double x, double y)	
long isgreater (doublen x, doublen y)		long isordered (doublen x, doublen y)	
int isgreater (half x, half y)		int isordered (half x, half y)	
short isgreater (halfn x, halfn y)	Compare of x >= y	int isunordered (float x, float y)	Test if arguments are unordered
intn isgreater (floatn x, floatn y)		intn isunordered (floatn x, floatn y)	
double isgreater (double x, double y)		double isunordered (double x, double y)	
long isgreater (doublen x, doublen y)		long isunordered (doublen x, doublen y)	
int isgreater (half x, half y)		int isunordered (half x, half y)	
short isgreater (halfn x, halfn y)	Compare of x < y	int isignbit (float)	Test for sign bit
intn isless (floatn x, floatn y)		intn isignbit (floatn)	
double isless (double x, double y)		double isignbit (double)	
long isless (doublen x, doublen y)		long isignbit (doublen)	
int isless (half x, half y)		int isignbit (half)	
short isless (halfn x, halfn y)	Compare of x <= y	int any (S x)	1 if MSB in any component of x is set; else 0
intn islesseq (floatn x, floatn y)		intn any (S x)	
double islesseq (double x, double y)		double any (S x)	
long islesseq (doublen x, doublen y)		long any (S x)	
int islesseq (half x, half y)		int any (S x)	
short islesseq (halfn x, halfn y)	Compare of (x < y)    (x > y)	T bselect (T a, T b, T c)	Each bit of result is corresponding bit of a if corresponding bit of c is 0
intn islesseq (floatn x, floatn y)		halfn bselect (halfn a, halfn b, halfn c)	
double islesseq (double x, double y)		double bselect (double a, double b, double c)	
long islesseq (doublen x, doublen y)		T select (T a, T b, S c)	
int islesseq (half x, half y)		double select (double, double, long)	
short islesseq (halfn x, halfn y)	double select (double, double, ulong)	For each component of a vector type, result() = if MSB of c[] is set ? b[] : a[] for scalar type, result = c ? b : a	
int isless (float x, float y)	half select (half, half, ushort)		
intn isless (floatn x, floatn y)			
double isless (double x, double y)			
long isless (doublen x, doublen y)			
int isless (half x, half y)	Test for finite value	int isfinite (float)	
intn isless (floatn x, floatn y)		intn isfinite (floatn)	
double isless (double x, double y)		double isfinite (double)	
long isless (doublen x, doublen y)		long isfinite (doublen)	
int isless (half x, half y)		int isfinite (half)	
short isless (halfn x, halfn y)		short isfinite (halfn)	

Vector Data Load/Store Functions [6.11.7]	
<p>Q is an Address Space Qualifier listed in 6.5 unless otherwise noted. # defaults to the current rounding mode, or is one of the Rounding Modes listed in 6.2.3.2. T is type char, uchar, short, ushort, int, sint, long, ulong, half, or float (or optionally double). Tn refers to the vector form of type T. Optional extensions enable the double_n, double_n, half, and half_n types.</p>	
void vload (size_t offset, const Q T *p)	Read vector data from memory
void vstore (Tn data, size_t offset, Q T *p)	Write vector data to memory (Q in the function cannot be __constant)
float vload_half (size_t offset, const Q half *p)	Read a half from memory
floatn vload_half (size_t offset, const Q half *p)	Read multiple halves from memory
void vstore_half (float data, size_t offset, Q half *p)	Write a half to memory
void vstore_half_R (float data, size_t offset, Q half *p)	(Q in the function cannot be __constant)
void vstore_halfn (floatn data, size_t offset, Q half *p)	Write a half vector to memory
void vstore_halfn_R (floatn data, size_t offset, Q half *p)	(Q in the function cannot be __constant)
floatn vload_halfn (size_t offset, const Q half *p)	sizeof (floatn) bytes of data read from location (p + (offset * n))
void vstore_halfn (floatn data, size_t offset, Q half *p)	Write a half vector to vector aligned memory
void vstore_halfn_R (floatn data, size_t offset, Q half *p)	(Q in the function cannot be __constant)

Atomic Functions [6.11.11.9.4]		Atomic add, and store	
<p>T is type int or unsigned int. T may also be type float for atomic_xchg, and type long or ulong for volatile_64-bit atomic functions. Q is volatile __global or volatile __local, except Q must be volatile __global for atomic_xchg when T is float.</p>			
<p>The built-in atomic functions for 32-bit values begin with atomic_ while the extended 64-bit atomic functions begin with atom_ for example:</p>			
Built-in atomic function	Extended atomic function	Atomic_add (Q T *p, T val)	Read, add, and store
atomic_add (Q T *p)	atom_add (Q T *p)	Atomic_sub (Q T *p, T val)	Read, subtract, and store
		Atomic_xchg (Q T *p, T val)	Read, swap, and store
		Atomic_inc (Q T *p)	Read, increment, and store
		Atomic_dec (Q T *p)	Read, decrement, and store
		Atomic_cmpxchg (Q T *p, T cmp, T val)	Read and store (*p == cmp) ? val : *p
		Atomic_min (Q T *p, T val)	Read, store min(*p, val)
		Atomic_max (Q T *p, T val)	Read, store max(*p, val)
		Atomic_and (Q T *p, T val)	Read, store (*p & val)
		Atomic_or (Q T *p, T val)	Read, store (*p   val)
		Atomic_xor (Q T *p, T val)	Read, store (*p ^ val)

Async Copies and Prefetch Functions [6.11.10]	
<p>T is type char, charn, uchar, uchar_n, short, shorts, ushort, ushortn, int, intn, sint, sintn, long, longn, ulong, ulongn, float, floatn, and optionally half double, double_n, and double_n types.</p>	
event_t async_work_group_copy (local T *dst, const __global T *src, size_t num_pertypes, event_t event)	Copies num_pertypes T elements from src to dst
event_t async_work_group_strided_copy (local T *dst, const __global T *src, size_t num_pertypes, size_t src_stride, event_t event)	Copies num_pertypes T elements from src to dst
void wait_group_events (int num_events, event_t *event_list)	Wait for events that identify the async_work_group_copy operations to complete
void prefetch (const __global T *p, size_t num_pertypes)	Prefetch num_pertypes * sizeof(T) bytes into the global cache

- Page 5

# Quick Reference Card

## OpenCL API 1.1 Quick Reference Card - Page 5

### Miscellaneous Vector Built-In Functions [6.11.12]

*in* and *in* mean the 2, 4, 8, or 16-component vectors of *char*, *uchar*, *short*, *ushort*, *half*, *int*, *uint*, *long*, *ulong*, *float*, *double*. *in* means the built-in unsigned integer data types. For *vec\_step*, it also includes *char*, *uchar*, *short*, *ushort*, *half*, *int*, *uint*, *long*, *ulong*, *float*, and *double*. Half and double types are enabled by *cl\_khr\_fp16* and *cl\_khr\_fp64* respectively.

<code>int vec_step (in c)</code> <code>int vec_step (opencl)</code>	Takes a built-in scalar or vector data type argument and returns an integer value representing the number of elements in the scalar or vector.	<code>in shuffle (in x, in mask)</code> <code>in shuffle2 (in x, in y, in mask)</code>	Construct permutation of elements from one or two input vectors, return a vector with same element type as input & length that is the same as the shuffle mask.
--	--	---	---

### Synchronization, Explicit Mem. Fence [6.11.9-10]

*flops* argument is the memory address space, set to a combination of *CLK\_LOCAL\_MEM\_FENCE* and *CLK\_GLOBAL\_MEM\_FENCE*.

<code>void barrier (cl_mem_fence_flags flops)</code>	All work-items in a work-group must execute this before any can continue.
<code>void mem_fence (cl_mem_fence_flags flops)</code>	Orders loads and stores of a work-item executing a kernel.
<code>void read_mem_fence (cl_mem_fence_flags flops)</code>	Orders memory loads.
<code>void write_mem_fence (cl_mem_fence_flags flops)</code>	Orders memory stores.

**OpenCL Graphics:** Following is a subset of the OpenCL API specification that pertains to graphics.

### Image Read and Write Built-in Functions [6.11.13, 9.1, 9.6.4]

The built-in functions defined in this section can only be used with image memory objects created with `clCreateImage2D` or `clCreateImage3D`. *sampler* specifies the addressing and filtering mode to use. *in* is to enable read, *imageh* and *write\_imageh* enable extension of *cl\_khr\_fp16*. *in* is to enable type `image3d_t` in `write_image3d_t`, enable extension `cl_khr_3d_image_writes`.

<code>float4 read_imagef (image2d_t image, sampler_t sampler, int2 coord)</code> <code>float4 read_imagef (image2d_t image, sampler_t sampler, float2 coord)</code> <code>int4 read_imagei (image2d_t image, sampler_t sampler, int2 coord)</code> <code>uint4 read_imageui (image2d_t image, sampler_t sampler, int2 coord)</code> <code>half4 read_imageh (image2d_t image, sampler_t sampler, int2 coord)</code> <code>half4 read_imageh (image2d_t image, sampler_t sampler, float2 coord)</code>	Read an element from a 2D image.	<code>uint4 read_imageui (image3d_t image, sampler_t sampler, int4 coord)</code> <code>uint4 read_imageui (image3d_t image, sampler_t sampler, float4 coord)</code> <code>int4 read_imagei (image3d_t image, sampler_t sampler, int4 coord)</code> <code>int4 read_imagei (image3d_t image, sampler_t sampler, float4 coord)</code>	Read an element from a 3D image.
<code>void write_imagef (image2d_t image, int2 coord, float4 color)</code> <code>void write_imageui (image2d_t image, int2 coord, uint4 color)</code> <code>void write_imageh (image2d_t image, int2 coord, half4 color)</code>	Write color value to (x, y) location specified by coord in the 2D image.	<code>int4 get_image_width (image2d_t image)</code> <code>int4 get_image_height (image2d_t image)</code> <code>int4 get_image_depth (image3d_t image)</code> <code>int4 get_image_channel_data_type (image2d_t image)</code> <code>int4 get_image_channel_data_type (image3d_t image)</code> <code>int4 get_image_channel_order (image2d_t image)</code> <code>int4 get_image_channel_order (image3d_t image)</code> <code>int4 get_image_dim (image2d_t image)</code> <code>int4 get_image_dim (image3d_t image)</code>	Image width in pixels Image height in pixels Image depth in pixels Image channel data type Image channel order Image width, height, and depth Image width, height, and depth
<code>float4 read_imagef (image3d_t image, sampler_t sampler, int4 coord)</code> <code>float4 read_imagef (image3d_t image, sampler_t sampler, float4 coord)</code> <code>int4 read_imagei (image3d_t image, sampler_t sampler, int4 coord)</code> <code>int4 read_imagei (image3d_t image, sampler_t sampler, float4 coord)</code>	Read an element from a 3D image.	Use this pragma to enable type <code>image3d_t</code> in <code>write_image3d_t</code> : <code>#pragma OPENCL_EXTENSION cl_khr_3d_image_writes : enable</code>	Writes color at coord in the 3D image.

### Image Objects

#### Create Image Objects [5.3.1]

`cl_mem clCreateImage2D (cl_context context, cl_mem_flags flags, const cl_image_format *image_format, size_t image_width, size_t image_height, size_t image_row_pitch, void *host_ptr, cl_int *errcode_ret) flops;` (also for `clCreateImage3D`, `clGetSupportedImageFormats`)  
*CL\_MEM\_READ\_WRITE*, *CL\_MEM\_READ\_ONLY*, *CL\_MEM\_WRITE\_ONLY*, *CL\_MEM\_ALLOC\_COPY\_HOST\_PTR*

`cl_mem clCreateImage3D (cl_context context, cl_mem_flags flags, const cl_image_format *image_format, size_t image_width, size_t image_height, size_t image_depth, size_t image_row_pitch, size_t image_slice_pitch, void *host_ptr, cl_int *errcode_ret) flops;` See `clCreateImage2D`

#### Query List of Supported Image Formats [5.3.2]

`cl_int clGetSupportedImageFormats (cl_context context, cl_mem_flags flags, cl_mem_object_type image_type, cl_uint num_entries, cl_image_format *image_formats, cl_uint *num_image_formats) flops;` See `clCreateImage2D`

#### Copy Between Image, Buffer Objects [5.3.4]

`cl_int clEnqueueCopyImageToBuffer (cl_command_queue command_queue, cl_mem src_image, cl_mem dst_buffer, const size_t src_origin[3], const size_t region[3], size_t dst_offset, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

`cl_int clEnqueueCopyBufferToImage (cl_command_queue command_queue, cl_mem src_buffer, cl_mem dst_image, size_t src_offset, const size_t dst_origin[3], const size_t region[3], cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

#### Map and Unmap Image Objects [5.3.5]

`void * clEnqueueMapImage (cl_command_queue command_queue, cl_mem image, cl_bool blocking_map, cl_map_flags map_flags, const size_t origin[3], const size_t region[3], size_t *row_pitch, size_t *slice_pitch, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event, cl_int *errcode_ret)`

### Read, Write, Copy Image Objects [5.3.3]

`cl_int clEnqueueReadImage (cl_command_queue command_queue, cl_mem image, cl_bool blocking_read, const size_t origin[3], const size_t region[3], size_t row_pitch, size_t slice_pitch, void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

`cl_int clEnqueueWriteImage (cl_command_queue command_queue, cl_mem image, cl_bool blocking_write, const size_t origin[3], const size_t region[3], size_t input_row_pitch, size_t input_slice_pitch, const void *ptr, cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

`cl_int clEnqueueCopyImage (cl_command_queue command_queue, cl_mem src_image, cl_mem dst_image, const size_t src_origin[3], const size_t dst_origin[3], const size_t region[3], cl_uint num_events_in_wait_list, const cl_event *event_wait_list, cl_event *event)`

#### Query Image Objects [5.3.4]

`cl_int clGetMemObjectInfo (cl_mem memobj, cl_mem_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret) param_name: CL_MEM_TYPE, FLAGS, SIZE_HOST_PTR, CL_MEM_MAP_REFERENCE_COUNT, CL_MEM_CONTEXT_OFFSET, CL_MEM_ASSOCIATED_MEMOBJECT`

`cl_int clGetImageInfo (cl_mem image, cl_image_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret) param_name: CL_IMAGE_FORMAT_ELEMENT_SIZE, CL_IMAGE_ROW_SLICE_PITCH, CL_IMAGE_HEIGHT, WIDTH, DEPTH, CL_IMAGE_DSSO, SUBRESOURCE_KHR, CL_MEM_DSSO_RESOURCE_KHR`

#### Access Qualifiers [6.4]

Apply to image `image2d_t` and `image3d_t` types to declare if the image memory object is being read or written by a kernel. The default qualifier is `_read_only`.  
`_read_only`, `_write_only`

### Image Formats [5.3.1.1, 5.5]

Supported image formats: `Image_channel_order` with `image_channel_data_type`.

Built-in support: (Table 5.7)  
`CL_RGBA`: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

`CL_BGRA`: `CL_UNORM_INT8`

Optional support: (Table 5.8)

`CL_R`, `CL_A`: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

`CL_INTENSITY`: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`

`CL_LUMINANCE`: `CL_UNORM_INT8`, `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

`CL_RGB`, `CL_RGBA`: `CL_HALF_FLOAT`, `CL_FLOAT`, `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_UNSIGNED_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

`CL_RGBA`: `CL_UNORM_SHORT_555`, `CL_UNSIGNED_INT_101010`

`CL_RGBA`: `CL_UNORM_INT8`, `CL_SIGNED_INT8`, `CL_UNSIGNED_INT8`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`, `CL_SIGNED_INT16`, `CL_UNSIGNED_INT16`

### Sampler Objects [5.9]

`cl_sampler clCreateSampler (cl_context context, cl_bool normalized_coords, cl_addressing_mode addressing_mode, cl_filter_mode filter_mode, cl_int *errcode_ret)`

`cl_int clRetainSampler (cl_sampler sampler)`

`cl_int clReleaseSampler (cl_sampler sampler)`

`cl_int clGetSamplerInfo (cl_sampler sampler, cl_sampler_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret)`

param\_name: `CL_SAMPLER_REFERENCE_COUNT`, `CL_SAMPLER_CONTEXT_FILTER_MODE`, `CL_SAMPLER_ADDRESSING_MODE`, `CL_SAMPLER_NORMALIZED_COORDS`

# Quick Reference Card

- Page 6
- Only touches on C++ bindings, etc.
- Necessary and inevitable consequence of generality.
- But works, and works a lot of places

## OpenCL API 1.1 Quick Reference Card - Page 6

### Sampler Declaration Fields [5.11.1.1]

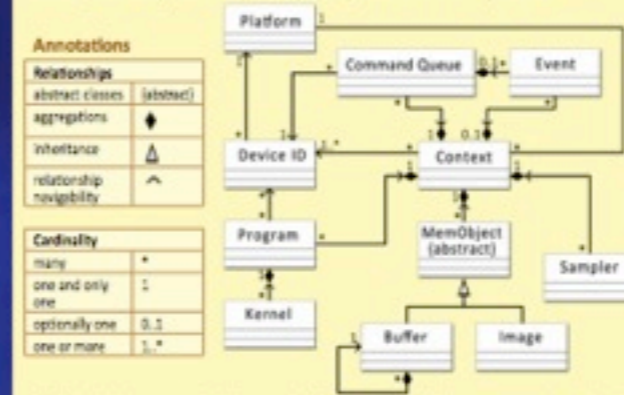
The sampler can be passed as an argument to the kernel using `dSetKernelArg`, or it can be a constant variable of type `sampler_t` declared in the program source.

```
const sampler_t <sampler_name> =
  <normalized-mode> | <address-mode> | <filter-mode>
normalized-mode:
  CLK_NORMALIZED_COORDS_(TRUE, FALSE)
```

```
address-mode:
  CLK_ADDRESS_(REPEAT, CLAMP, NONE),
  CLK_ADDRESS_(CLAMP_TO_EDGE, MIRRORED_REPEAT)
filter-mode:
  CLK_FILTER_NEAREST, CLK_FILTER_LINEAR
```

### OpenCL Class Diagram [5.13]

The figure below describes the OpenCL specification as a class diagram using the Unified Modeling Language<sup>1</sup> (UML) notation. The diagram shows both nodes and edges which are classes and their relationships. As a simplification it shows only classes, and no attributes or operations.



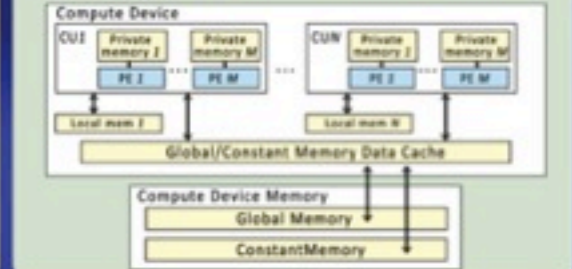
<sup>1</sup> Unified Modeling Language (<http://www.uml.org/>) is a trademark of Object Management Group (OMG).

### OpenCL Device Architecture Diagram [3.1]

The table below shows memory regions with allocation and memory access capabilities.

	Global	Constant	Local	Private
Host	Dynamic allocation Read/Write access	Dynamic allocation Read/Write access	Dynamic allocation No access	No allocation No access
Kernel	No allocation Read/Write access	Static allocation Read-only access	Static allocation Read/Write access	Static allocation Read/Write access

This conceptual OpenCL device architecture diagram shows processing elements (PE), compute units (CU), and devices. The host is not shown.



### OpenCL/OpenGL Sharing APIs

Creating OpenCL memory objects from OpenGL objects using `dCreateFromGLBuffer`, `dCreateFromGLTexture2D`, `dCreateFromGLTexture3D`, and `dCreateFromGLRenderbuffer` ensure that the storage of the OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

**CL Buffer Objects > GL Buffer Objects [3.8.2]**  
`d_mem dCreateFromGLBuffer (d_context context, d_mem_flags flags, GLuint bufobj, int *errcode_ret) flags: CL_MEM_READ_WRITE | ONLY_CL_MEM_READ_WRITE`

**CL Image Objects > GL Textures [3.8.3]**  
`d_mem dCreateFromGLTexture2D (d_context context, d_mem_flags flags, GLenum texture_target, GLuint miplevel, GLuint texture, d_int *errcode_ret) flags: See dCreateFromGLBuffer texture_target: GL_TEXTURE_2D, RECTANGLE, GL_TEXTURE_CUBE_MAP_POSITIVE_X, Y, Z, GL_TEXTURE_CUBE_MAP_NEGATIVE_X, Y, Z`

`d_mem dCreateFromGLTexture3D (d_context context, d_mem_flags flags, GLenum texture_target, GLuint miplevel, GLuint texture, d_int *errcode_ret) flags: See dCreateFromGLBuffer texture_target: GL_TEXTURE_3D`

### CL Image Objects > GL Renderbuffers [3.8.4]

`d_mem dCreateFromGLRenderbuffer (d_context context, d_mem_flags flags, GLuint renderbuffer, d_int *errcode_ret) flags: dCreateFromGLBuffer`

**Query Information [3.8.5]**  
`d_int dGetGLObjectInfo (d_mem memobj, d_gl_object_type *gl_object_type, GLuint *gl_object_name) *gl_object_type returns: CL_GL_OBJECT_BUFFER, CL_GL_OBJECT_TEXTURE2D, TEXTURE3D, CL_GL_OBJECT_RENDERBUFFER`

`d_int dGetGLTextureInfo (d_mem memobj, d_gl_texture_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret) param_name: CL_GL_TEXTURE_TARGET, CL_GL_MIPMAP_LEVEL`

### Share Objects [3.8.6]

`d_int dEnqueueAcquireGLObjects (d_command_queue command_queue, d_uint num_objects, const d_mem *mem_objects, d_uint num_events_in_wait_list, const d_event *event_wait_list, d_event *event)`

`d_int dEnqueueReleaseGLObjects (d_command_queue command_queue, d_uint num_objects, const d_mem *mem_objects, d_uint num_events_in_wait_list, const d_event *event_wait_list, d_event *event)`

**CL Event Objects > GL Sync Objects [3.9]**  
`d_event dCreateEventFromGLSyncObj (d_context context, GLsync sync, d_int *errcode_ret)`

**CL Context > GL Context, Sharegroup [3.1]**  
`d_int dGetGLContextInfoKHR (const d_context_properties *properties, d_gl_context_info param_name, size_t param_value_size, void *param_value, size_t *param_value_size_ret) param_name: CL_DEVICES_FOR_GL_CONTEXT_KHR, CL_CURRENT_DEVICE_FOR_GL_CONTEXT_KHR`

### OpenCL/Direct3D 10 Sharing APIs [3.10]

Creating OpenCL memory objects from OpenGL objects using `dCreateFromGLBuffer`, `dCreateFromGLTexture2D`, `dCreateFromGLTexture3D`, or `dCreateFromGLRenderbuffer` ensures that the storage of that OpenGL object will not be deleted while the corresponding OpenCL memory object exists.

`d_int dGetDeviceIDFromD3D10KHR (d_platform_id platform, d_d3d10_device_source_khr d3d_device_source, void *obj, d_d3d10_device_set_khr d3d_device_set, d_uint num_entries, d_device_id *devices, d_sint *num_devices) d3d_device_source: CL_D3D10_DEVICE_KHR, CL_D3D10_DDI_ADAPTER_KHR d3d_device_set: CL_ALL_DEVICES_FOR_D3D10_KHR, CL_PREFERRED_DEVICES_FOR_D3D10_KHR`

`d_mem dCreateFromD3D10BufferKHR (d_context context, d_mem_flags flags, ID3D10Buffer *resource, d_int *errcode_ret) flags: CL_MEM_READ_WRITE | ONLY_CL_MEM_READ_WRITE`

`d_mem dCreateFromD3D10Texture2DKHR (d_context context, d_mem_flags flags, ID3D10Texture2D *resource, UINT subresource, d_int *errcode_ret) flags: See dCreateFromD3D10BufferKHR`

`d_mem dCreateFromD3D10Texture3DKHR (d_context context, d_mem_flags flags, ID3D10Texture3D *resource, UINT subresource, d_int *errcode_ret) flags: See dCreateFromD3D10BufferKHR`

`d_int dEnqueueAcquireD3D10ObjectsKHR (d_command_queue command_queue, d_uint num_objects, const d_mem *mem_objects, d_uint num_events_in_wait_list, const d_event *event_wait_list, d_event *event)`

`d_int dEnqueueReleaseD3D10ObjectsKHR (d_command_queue command_queue, d_uint num_objects, const d_mem *mem_objects, d_uint num_events_in_wait_list, const d_event *event_wait_list, d_event *event)`



The Khronos Group is an industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics and dynamic media on a wide variety of platforms and devices. See [www.khronos.org](http://www.khronos.org) to learn more about the Khronos Group.

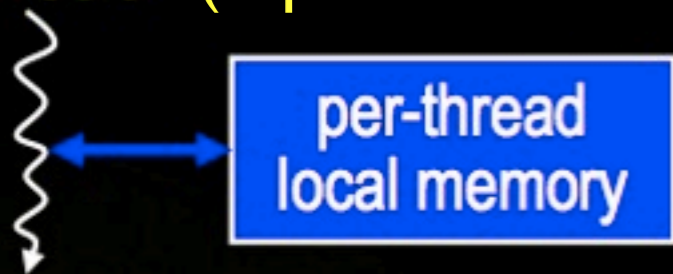
OpenCL is a trademark of Apple Inc. and is used under license by Khronos.



# CUDA In One Slide (And OpenCL)

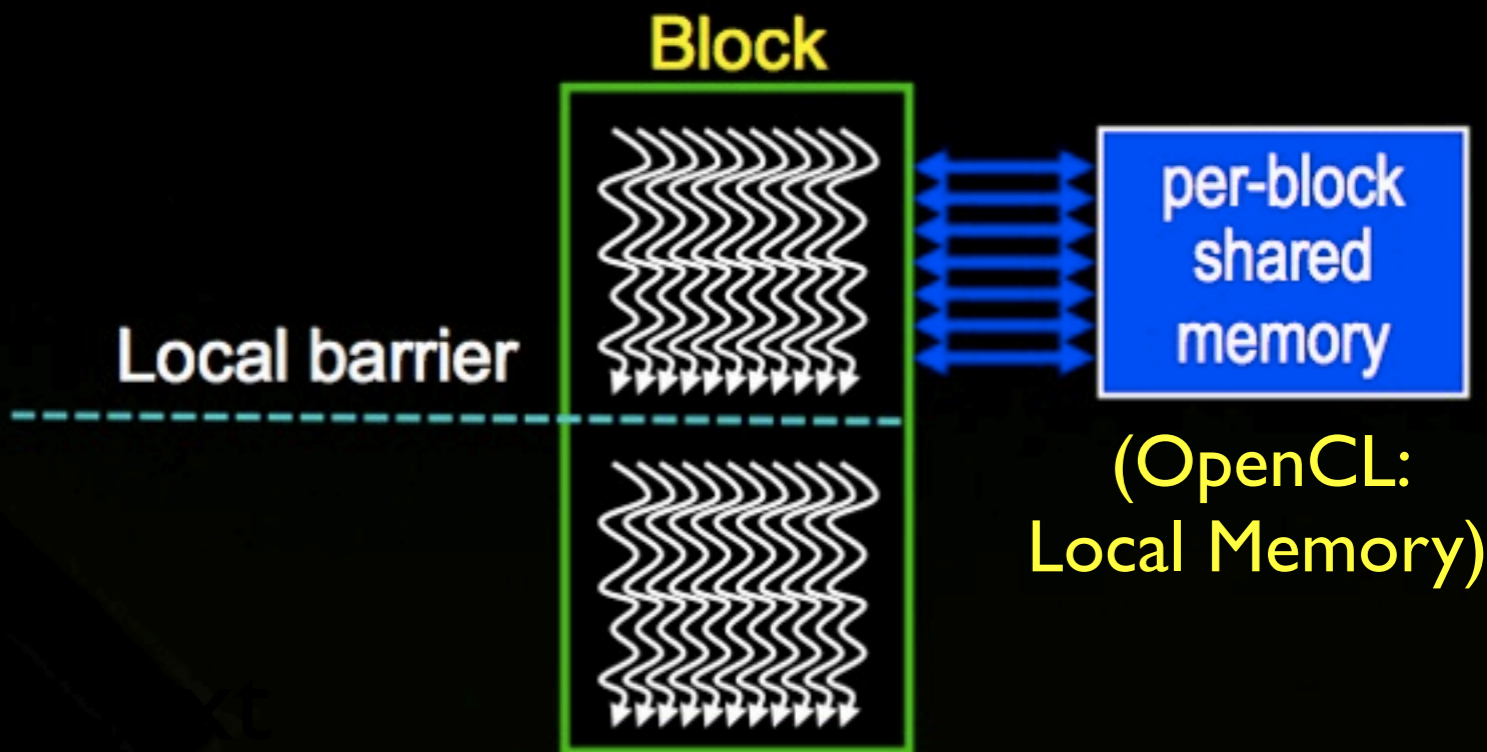


Thread (OpenCL: Work Item)

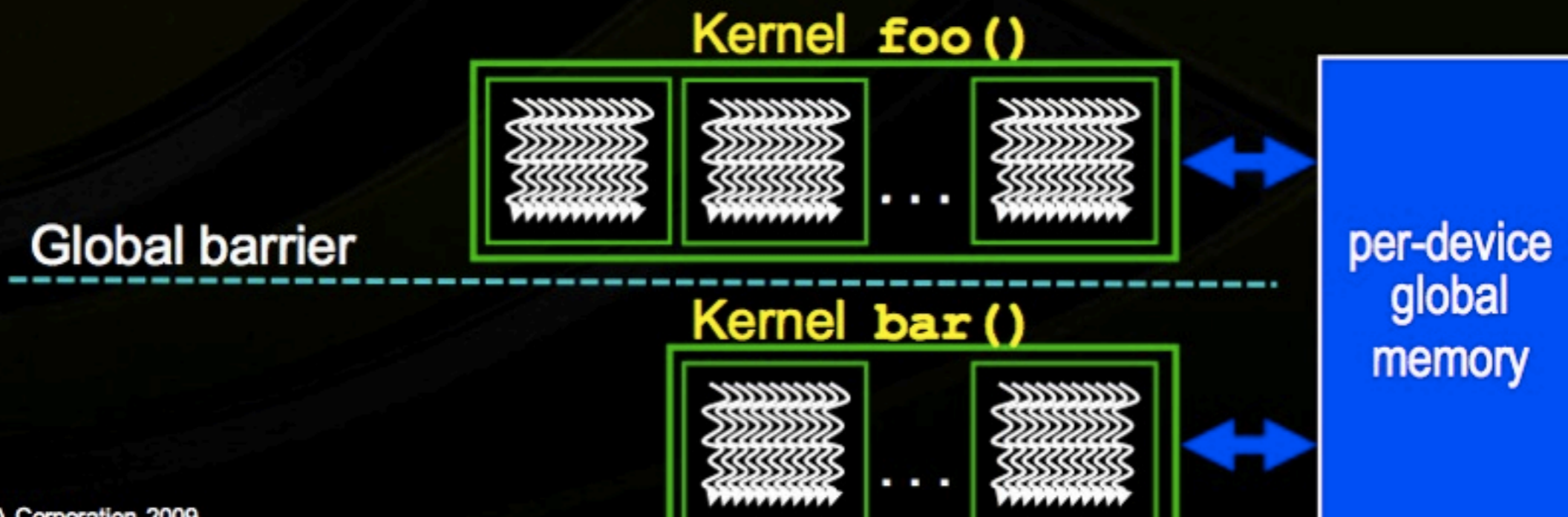


(OpenCL:  
Private Memory)

(OpenCL: Work Group)



(OpenCL:  
Local Memory)



# CUDA vs OpenCL terminology

- Since maps to similar hardware, basic concepts the same
- Some terminology changes; some better, some worse.

CUDA	OpenCL
Thread	Work Item
Block	Work Group
Device Mem	Global Mem
Constant Mem	Constant Mem
Shared Mem	Local Mem
Local Mem	Private Mem

# CUDA vs OpenCL kernel code

- Since maps to similar hardware, basic concepts the same
- Some terminology changes; some better, some worse.

CUDA	OpenCL
<code>__global__</code>	<code>__kernel</code>
<code>__device__</code> (function)	
<code>__constant__</code>	<code>__constant</code>
<code>__device__</code> (mem)	<code>__global</code>
<code>__shared__</code>	<code>__local</code>
Local Mem	Private Mem
<code>__syncthreads()</code>	<code>barrier()</code>

# CUDA vs OpenCL kernel code

- Functions rather than variables for place in grid
- “global work size” and “local work size” vs  $\text{blockDim} * \text{gridDim}$  and  $\text{BlockDim}$
- `get_global_id()` calculates global position for you

CUDA	OpenCL
<code>gridDim</code>	<code>get_num_groups()</code>
<code>blockDim</code>	<code>get_local_size()</code>
<code>blockIdx</code>	<code>get_group_id()</code>
<code>threadIdx</code>	<code>get_local_id()</code>
-	<code>get_global_id()</code>
-	<code>get_global_size()</code>

```
__global__ void cuda_saxpb(const float *xd,
                          const float a,
                          const float b,
                          float *yd, const int n) {

    int i = threadIdx.x + blockIdx.x*blockDim.x;
    if (i<n) {
        yd[i] = a*xd[i]+b;
    }
    return;
}
```

CUDA

---

```
__kernel void openc1_saxpb(__global const float *x,
                          const float a, const float b,
                          __global float *y)
{
    int i = get_global_id(0);
    if (i < get_global_size(0) )
        y[i] = a*x[i] + b;
}
```

OpenCL



# CUDA

```
/* run GPU code */
CHK_CUDA( cudaMalloc(&xd, n*sizeof(float)) );
CHK_CUDA( cudaMalloc(&yd, n*sizeof(float)) );

tick(&gputimer);
CHK_CUDA( cudaMemcpy(xd, x, n*sizeof(float), cudaMemcpyHostToDevice) );

blocksize = (n+nblocks-1)/nblocks;
for (i=0; i<niters; i++) {
    cuda_saxpb<<<nblocks, blocksize>>>(xd, a, b, yd, n);
}
CHK_CUDA( cudaMemcpy(ycuda, yd, n*sizeof(float), cudaMemcpyDeviceToHost) );
gputime = tock(&gputimer);

CHK_CUDA( cudaFree(xd) );
CHK_CUDA( cudaFree(yd) );
```

```

/* create OpenCL device & context */
cl_platform_id clPlatform;
err = clGetPlatformIDs(1, &clPlatform, NULL);
chk(err, "Get Platform");

/* query all devices available to the context */

cl_device_id device;
err = clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
chk(err, "Get Device IDs");

cl_context hContext;
hContext = clCreateContext(0, 1, &device, NULL, NULL, &err);
chk(err, "Get Context");

/* create a command queue for first device the context reported */
cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, device, 0, &err);
chk(err, "Create Queue");

/* create & compile program */
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, nlines, kernelsrc, 0, &err);
chk(err, "Create Program");

err = clBuildProgram(hProgram, 1, &device, NULL, NULL, NULL);
buildchk(err, "Build Program");

/* create kernel */
cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, "opencl_saxpb", &err);
chk(err, "Create Kernel");

/* allocate device memory */
cl_mem yd;
cl_mem xd;
tick(&gputimer);
xd = clCreateBuffer(hContext,
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    x,
    &err);
chk(err, "Create xd");

```

```

yd = clCreateBuffer(hContext,
    CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    yopencl,
    &err);
chk(err, "Create yd");

/* setup parameter values */
err = clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&xd);
err |= clSetKernelArg(hKernel, 1, sizeof(cl_float), (void *)&a);
err |= clSetKernelArg(hKernel, 2, sizeof(cl_float), (void *)&b);
err |= clSetKernelArg(hKernel, 3, sizeof(cl_mem), (void *)&yd);
chk(err, "Set args");

/* execute kernel */
const size_t ksize=n;
const size_t kblocksize=blocksize;
err = clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0,
    &ksize, &kblocksize, 0, 0, 0);
chk(err, "Enqueue Kernel");

// copy results from device back to host
err = clEnqueueReadBuffer(hCmdQueue, yd, CL_TRUE, 0,
    n * sizeof(cl_float),
    yopencl, 0, 0, 0);
chk(err, "Enqueue Read");

clReleaseMemObject(xd);
clReleaseMemObject(yd);
clReleaseProgram(hProgram);
clReleaseKernel(hKernel);
clReleaseCommandQueue(hCmdQueue);
clReleaseContext(hContext);

```

# OpenCL

# More general - more verbose

- CUDA makes simple things simple - easy to do with fixed view of hardware
- OpenCL doesn't. But more complicated situations are exactly the same; learning curve steep, but short.
- Let's step through this (new OpenCL 1.1 C++ bindings a little nicer)

```
/* create OpenCL device & context */
cl_platform_id clPlatform;
err = clGetPlatformIDs(1, &clPlatform, NULL);
chk(err, "Get Platform");

/* query all devices available to the context */
cl_device_id device;
err = clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
chk(err, "Get Device IDs");

cl_context hContext;
hContext = clCreateContext(0, 1, &device, NULL, NULL, &err);
chk(err, "Get Context");

/* create a command queue for first device the context reported */
cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, device, 0, &err);
chk(err, "Create Queue");

/* create & compile program */
cl_program hProgram;
hProgram = clCreateProgramWithSource(hContext, nlines, kernelsrc, 0, &err);
chk(err, "Create Program");

err = clBuildProgram(hProgram, 1, &device, NULL, NULL, NULL);
buildchk(err, "Build Program");

/* create kernel */
cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, "openc1_saxpb", &err);
chk(err, "Create Kernel");

/* allocate device memory */
cl_mem yd;
cl_mem xd;
tick(&gputimer);
xd = clCreateBuffer(hContext,
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    x,
    &err);
chk(err, "Create xd");
```

```
yd = clCreateBuffer(hContext,
    CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    yopenc1,
    &err);
chk(err, "Create yd");

/* setup parameter values */
err = clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *) yd);
err |= clSetKernelArg(hKernel, 1, sizeof(cl_float), (void *) 1);
err |= clSetKernelArg(hKernel, 2, sizeof(cl_float), (void *) 1);
err |= clSetKernelArg(hKernel, 3, sizeof(cl_mem), (void *) xd);
chk(err, "Set args");

/* execute kernel */
const size_t ksize=n;
const size_t kblocksize=blocksize;
err = clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0,
    &ksize, &kblocksize, 0, 0, 0);
chk(err, "Enqueue Kernel");

// copy results from device back to host
err = clEnqueueReadBuffer(hCmdQueue, yd, CL_TRUE, 0,
    n * sizeof(cl_float),
    yopenc1, 0, 0, 0);
chk(err, "Enqueue Read");

clReleaseMemObject(xd);
clReleaseMemObject(yd);
clReleaseProgram(hProgram);
clReleaseKernel(hKernel);
clReleaseCommandQueue(hCmdQueue);
clReleaseContext(hContext);
```



Have to do something like this in CUDA for multiple GPUs

```
/* create OpenCL device & context */
cl_platform_id clPlatform;
err = clGetPlatformIDs(1, &clPlatform, NULL);
chk(err, "Get Platform");

/* query all devices available to the context */
cl_device_id device;
err = clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
chk(err, "Get Device IDs");

cl_context hContext;
hContext = clCreateContext(0, 1, &device, NULL, NULL, &err);
chk(err, "Get Context");

/* create a command queue for first device the context reported */
cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, device, 0, &err);
chk(err, "Create Queue");
```

Doesn't have  
to be a GPU!  
"Host" could  
be a "device"

```
/* create OpenCL device & context */
cl_platform_id clPlatform;
err = clGetPlatformIDs(1, &clPlatform, NULL);
chk(err, "Get Platform");

/* query all devices available to the context */
cl_device_id device;
err = clGetDeviceIDs(clPlatform, CL_DEVICE_TYPE_GPU, 1, &device, NULL);
chk(err, "Get Device IDs");

cl_context hContext;
hContext = clCreateContext(0, 1, &device, NULL, NULL, &err);
chk(err, "Get Context");

/* create a command queue for first device the context reported */
cl_command_queue hCmdQueue;
hCmdQueue = clCreateCommandQueue(hContext, device, 0, &err);
chk(err, "Create Queue");
```

```
const char *kernelsrc[] = {
    "__kernel void opencl_saxpb(__global const float *x,",
    "    const float a, const float b,",
    "    __global float *y)",
    " { ",
    "    int i = get_global_id(0);",
    "    if (i < get_global_size(0) )",
    "        y[i] = a*x[i] + b;",
    " }"
};
const int nlines=8;
```

```
/* create & compile program */
cl_program hProgram,
hProgram = clCreateProgramWithSource(hContext, nlines, kernelsrc, 0, &err);
chk(err, "Create Program");

err = clBuildProgram(hProgram, 1, &device, NULL, NULL, NULL);
buildchk(err, "Build Program");

/* create kernel */
cl_kernel hKernel;
hKernel = clCreateKernel(hProgram, "opencl_saxpb", &err);
chk(err, "Create Kernel");
```

Have to compile,  
build program and  
identify kernels  
within the program!  
Allows real  
heterogeneity; also,  
run-time rebuilding  
of program  
(CUDA can now do  
this too)

As a result of run-time  
“device” compilation,  
build the main program  
just with normal compiler,  
link appropriate OpenCL  
libraries.

```
[ljdursi@tpb1 class5]$ make  
gcc -Wall -g -O3 -o block-saxpb block-saxpb.c -lOpenCL -lm  
[ljdursi@tpb1 class5]$ █
```

```

/* allocate device memory */
cl_mem yd;
cl_mem xd;
tick(&gputimer);
xd = clCreateBuffer(hContext,
    CL_MEM_READ_ONLY | CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    x,
    &err);
chk(err, "Create xd");

yd = clCreateBuffer(hContext,
    CL_MEM_COPY_HOST_PTR,
    n * sizeof(cl_float),
    yopencl,
    &err);
chk(err, "Create yd");

```

Malloc, Memcpy  
can be done in one  
step.

Can give many  
options (eg,  
READ\_ONLY for  
input values).



```
/* setup parameter values */  
err = clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&xd);  
err |= clSetKernelArg(hKernel, 1, sizeof(cl_float), (void *)&a);  
err |= clSetKernelArg(hKernel, 2, sizeof(cl_float), (void *)&b);  
err |= clSetKernelArg(hKernel, 3, sizeof(cl_mem), (void *)&y);  
chk(err, "Set args");
```

Have to manually marshal  
arguments to kernel launch.

```
err = clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0,  
                             &knsz, &kblocksize, 0, 0, 0);  
chk(err, "Enqueue Kernel");
```

```
/* setup parameter values */  
err = clSetKernelArg(hKernel, 0, sizeof(cl_mem), (void *)&xd);  
err |= clSetKernelArg(hKernel, 1, sizeof(cl_float), (void *)&a);  
err |= clSetKernelArg(hKernel, 2, sizeof(cl_float), (void *)&b);  
err |= clSetKernelArg(hKernel, 3, sizeof(cl_mem), (void *)&y);  
chk(err, "Set args");
```

Have to manually marshal  
arguments to kernel launch.

```
err = clEnqueueNDRangeKernel(hCmdQueue, hKernel, 1, 0,  
                             &knsz, &kblocksize, 0, 0, 0);  
chk(err, "Enqueue Kernel");
```

```
// copy results from device back to host
err = clEnqueueReadBuffer(hCmdQueue, yd, CL_TRUE, 0,
                          n * sizeof(cl_float),
                          yopencl, 0, 0, 0);
chk(err, "Enqueue Read");

clReleaseMemObject(xd);
clReleaseMemObject(yd);
clReleaseProgram(hProgram);
clReleaseKernel(hKernel);
clReleaseCommandQueue(hCmdQueue);
clReleaseContext(hContext);
```

Copy-back, cleanup not that  
different

---

Python + CUDA = PyCUDA

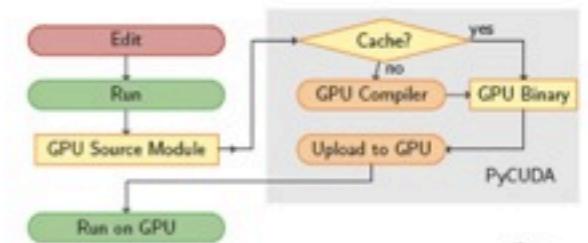
# Resources

- PyCUDA/PyOpenCL
- <http://mathematician.de/software/pycuda>
- Does a lot of the tedious initialization stuff for you
- Can focus on getting efficient kernels
- Can move to C/C++ later



- ▶ All of CUDA in a modern scripting language
- ▶ Full Documentation
- ▶ Free, open source (MIT)
- ▶ Also: PyOpenCL

- ▶ CUDA C Code = Strings
- ▶ Generate Code Easily
  - ▶ Automated Tuning
- ▶ Batteries included:  
GPU Arrays, RNG, ...
- ▶ Integration: numpy arrays,  
Plotting, Optimization, ...



# CUDA

```
import pycuda.driver as cuda
import pycuda.autoinit
from pycuda.compiler import SourceModule
import numpy

saxpb = SourceModule("""
__global__ void cuda_saxpb(const float *x, const float a,
                          const float b, float *y, const int n)
{
    int i = threadIdx.x + blockIdx.x*blockDim.x;
    if (i < n)
        y[i] = a*x[i] + b;
}
""")

npts = 1600
nblocks = 10
blocksize = (npts + nblocks-1)/nblocks
a = 5.
b = -1.
x = (numpy.random.randn(npts)).astype(numpy.float32)
y = numpy.empty_like(x)

# do GPU calculation

x_gpu = gpuarray.to_gpu(x)
y_gpu = gpuarray.to_gpu(y)

cuda_saxpb = saxpb.get_function(cuda_saxpb)
cuda_saxpb(x_gpu, a, b, y_gpu, npts, block=(blocksize,1,1))

# do CPU calculation
y = a * x - b

print 'GPU, CPU differ by ', numpy.sum(y_gpu - y)
```

# OpenCL

```
import pyopencl as cl
import numpy

ctx = cl.create_some_context()
queue = cl.CommandQueue(ctx)

prg = cl.Program(ctx, """
__kernel void cl_saxpb(__global const float *x,
                      const float a, const float b,
                      __global float *y)
{
    int i = get_global_id(0)
    if (i < get_global_size(0) )
        y[i] = a*x[i] + b;
}
""")

npts = 1600
nblocks = 10
blocksize = (npts + nblocks-1)/nblocks
a = 5.
b = -1.

x = (numpy.random.randn(npts)).astype(numpy.float32)
y = numpy.empty_like(x)

mf = cl.mem_flags
x_cl = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=x)
y_cl = cl.Buffer(ctx, mf.WRITE_ONLY, y.nbytes)

# do GPU calculation

prg.cl_saxpb(queue, x.shape, None, x_cl, a, b, y_cl)

y_gpu_res = numpy.empty_like(y)
cl.enqueue_read_buffer(queue, y_buf, y_gpu_res).wait()

# do CPU calculation
y = a * x - b

print 'GPU, CPU differ by ', numpy.sum(y_gpu_res - y)
```

# Resources

- NVidia ([http://www.nvidia.com/object/cuda\\_opengl\\_new.html](http://www.nvidia.com/object/cuda_opengl_new.html) )
- Mac Research ( <http://macresearch.org/opencv> )
- Apple (<http://developer.apple.com/> )
- Book ( <http://www.fixstars.com/en/company/books/opencv/> )
- AMD ( <http://developer.amd.com/zones/OpenCLZone> )

# Directives-Based GPGPU Programming: PGI Accelerator



# OpenACC

<http://www.openacc-standard.org/>

- CAPS, Cray, PGI, NVidia
- OpenMP for GPU
- Take existing proprietary compilers, find a minimal common feature set, implement
- Summer 2012

## General Syntax

**C**  
`#pragma acc directive [clause [[,] clause]...] new-line`

**FORTRAN**  
`!$acc directive [clause [[,] clause]...]`

Except for executable and declarative directives, OpenACC directives apply to the immediately following statement, loop or structured block.

## Parallel Construct

An accelerator **parallel** construct launches a number of gangs executing in parallel, where each gang may support multiple workers, each with vector or SIMD operations.

**C**  
`#pragma acc parallel [clause [[,] clause]...] new-line  
{ structured block }`

**FORTRAN**  
`!$acc parallel [clause [[,] clause]...]  
structured block  
!$acc end parallel`

Any data clause is allowed.

**OTHER CLAUSES**  
`if( condition )`

# PGI Accelerator

- Builds on strong OpenMP, vector performance
- On SciNet ARC:  
module load  
use.experimental pgi

```
!$acc region
  do k = 1,n1
    do i = 1,n3
      c(i,k) = 0.0
      do j = 1,n2
        c(i,k) = c(i,k) + a(i,j) * b(j,k)
      enddo
    enddo
  enddo
!$acc end region
```



<http://www.pgroup.com/resources/accel.htm>

# #pragma acc

- Define code region, data region, scheduling.
- In some cases, (esp Fortran) `#pragma acc region` + static analysis can do some pretty amazing things

```
32 !$acc region
33     do j=1,N
34         c(:,j) = 0.
35         do k=1,N
36             do i=1,N
37                 c(i,j) = c(i,j) + a(i,k)*b(k,j)
38             enddo
39         enddo
40     enddo
41 !$acc end region
```

```
$ pgf90 -Minfo=all,accel -fast -ta=nvidia,cc2.0 -o matrixf-gpu matrixf.f90
```

```
30, Memory zero idiom, array assignment replaced by call to pgf90_mzero8
31, Loop not vectorized/parallelized: too deeply nested
32, Generating copyout(c(1:n,1:n))
    Generating copyin(b(1:n,1:n))
    Generating copyin(a(1:n,1:n))
33, Loop is parallelizable
34, Loop is parallelizable
    Accelerator kernel generated
    33, !$acc do parallel, vector(16) ! blockidx%y threadidx%y
    34, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
    Generated 3 alternate versions of the loop
    Generated vector sse code for the loop
35, Loop carried reuse of 'c' prevents parallelization
36, Loop is parallelizable
    Accelerator kernel generated
    33, !$acc do parallel, vector(16) ! blockidx%y threadidx%y
    35, !$acc do seq(16)
        Cached references to size [16x16] block of 'a'
        Cached references to size [16x16] block of 'b'
    36, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
    Generated 3 alternate versions of the loop
    Generated vector sse code for the loop
    Generated 2 prefetch instructions for the loop
```

# Data clauses

- `#pragma acc copyin( )` - copies in array (or subarray) from host-kernel
- `copyout( )` - copies out data
- `copy( )` - in and out
- `local( )` - GPU-local array, doesn't need to be copied
- `region` - one or more kernels

```
$ pgf90 -Minfo=all,accel -fast -ta=nvidia,cc2.0 -o matrixf-gpu matrixf.f90
```

```
30, Memory zero idiom, array assignment replaced by call to pgf90_mzero8
31, Loop not vectorized/parallelized: too deeply nested
32, Generating copyout(c(1:n,1:n))
    Generating copyin(b(1:n,1:n))
    Generating copyin(a(1:n,1:n))
33, Loop is parallelizable
34, Loop is parallelizable
    Accelerator kernel generated
    33, !$acc do parallel, vector(16) ! blockidx%y threadidx%y
    34, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
    Generated 3 alternate versions of the loop
    Generated vector sse code for the loop
35, Loop carried reuse of 'c' prevents parallelization
36, Loop is parallelizable
    Accelerator kernel generated
    33, !$acc do parallel, vector(16) ! blockidx%y threadidx%y
    35, !$acc do seq(16)
        Cached references to size [16x16] block of 'a'
        Cached references to size [16x16] block of 'b'
    36, !$acc do parallel, vector(16) ! blockidx%x threadidx%x
    Generated 3 alternate versions of the loop
    Generated vector sse code for the loop
    Generated 2 prefetch instructions for the loop
```

# Data clauses

- `#pragma acc for parallel(N)` - breaks loop up between blocks, with blocksize N
- `#pragma acc for vector(N)` - parallelizes loop over threads in block, with vector size `Nregion` - one or more kernels
- `unroll` - force unrolling; `kernel` - force separate kernel



```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \  
        copyin(a[0:N-1][0:N-1]) \  
        copyout(maxchange)
```

```
    {
```

```
    for (int iter=0; iter<niters; iter++) {
```

```
        maxchange = 0.;
```

```
        {
```

```
        #pragma acc region
```

```
        for (int i=1; i<N-1; ++i)
```

```
            for (int j=1; j<N-1; ++j) {
```

```
                tmp[j][i] = w0*a[j][i] +
```

```
                    w1 * (a[j-1][i] + a[j][i-1] +
```

```
                        a[j+1][i] + a[j][i+1]) +
```

```
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +
```

```
                        a[j+1][i+1] + a[j-1][i+1]);
```

```
                double change = fabs(tmp[i][j] - a[i][j]);
```

```
                maxchange = fmax(maxchange, change);
```

```
            }
```

```
        #pragma acc region
```

```
        for (int i=0; i<N-1; ++i)
```

```
            for (int j=0; j<N-1; ++j)
```

```
                a[j][i] = tmp[j][i];
```

```
        }
```

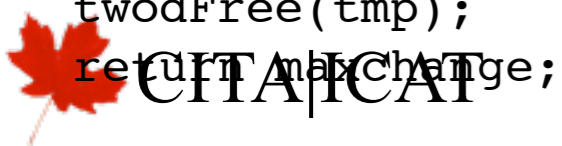
```
    }
```

```
    }
```

```
    twodFree(tmp);
```

```
    return maxchange;
```

```
}
```



```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \
        copyin(a[0:N-1][0:N-1]) \
        copyout(maxchange)
```

```
    {
    for (int iter=0; iter<niters; iter++) {
        maxchange = 0.;
        {
        #pragma acc region
        for (int i=1; i<N-1; ++i)
            for (int j=1; j<N-1; ++j) {
                tmp[j][i] = w0*a[j][i] +
                    w1 * (a[j-1][i] + a[j][i-1] +
                        a[j+1][i] + a[j][i+1]) +
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +
                        a[j+1][i+1] + a[j-1][i+1]);
                double change = fabs(tmp[i][j] - a[i][j]);
                maxchange = fmax(maxchange, change);
            }
        }
    }
}
```

```
    #pragma acc region
```

```
    for (int i=0; i<N-1; ++i)
```

```
        for (int j=0; j<N-1; ++j)
```

```
            a[j][i] = tmp[j][i];
```

```
    }
```

```
}
```

```
}
```

```
twodFree(tmp);
```

```
return maxchange;
```

```
}
```

All pointers  
need to be **restrict**

```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

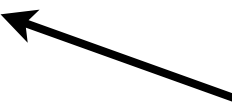
```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \  
                                copyin(a[0:N-1][0:N-1]) \  
                                copyout(maxchange)
```

```
    {  
    for (int iter=0; iter<niters; iter++) {  
        maxchange = 0.;  
        {  
        #pragma acc region  
        for (int i=1; i<N-1; ++i)  
            for (int j=1; j<N-1; ++j) {  
                tmp[j][i] = w0*a[j][i] +  
                    w1 * (a[j-1][i] + a[j][i-1] +  
                        a[j+1][i] + a[j][i+1]) +  
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +  
                        a[j+1][i+1] + a[j-1][i+1]);  
                double change = fabs(tmp[i][j] - a[i][j]);  
                maxchange = fmax(maxchange, change);  
            }  
        }  
    }  
    }  
}
```

A data region can  
contain multiple  
code regions



```
    #pragma acc region
```

```
    for (int i=0; i<N-1; ++i)
```

```
        for (int j=0; j<N-1; ++j)
```

```
            a[j][i] = tmp[j][i];
```

```
    }
```

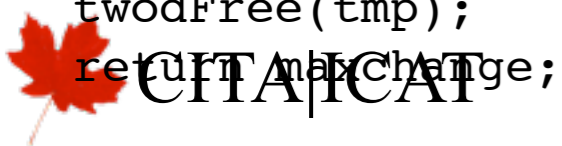
```
}
```

```
}
```

```
twodFree(tmp);
```

```
return maxchange;
```

```
}
```



```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \  
                                copyin(a[0:N-1][0:N-1]) \  
                                copyout(maxchange)
```

```
    {  
    for (int iter=0; iter<niters; iter++) {  
        maxchange = 0.;  
        {  
        #pragma acc region  
        for (int i=1; i<N-1; ++i)  
            for (int j=1; j<N-1; ++j) {  
                tmp[j][i] = w0*a[j][i] +  
                    w1 * (a[j-1][i] + a[j][i-1] +  
                        a[j+1][i] + a[j][i+1]) +  
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +  
                        a[j+1][i+1] + a[j-1][i+1]);  
                double change = fabs(tmp[i][j] - a[i][j]);  
                maxchange = fmax(maxchange, change); ←
```

This is recognized  
as a reduction and  
implemented

```
        #pragma acc region
```

```
        for (int i=0; i<N-1; ++i)
```

```
            for (int j=0; j<N-1; ++j)
```

```
                a[j][i] = tmp[j][i];
```

```
        }
```

```
    }
```

```
    }
```

```
    twodFree(tmp);
```

```
    return maxchange;
```

```
}
```

```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \  
        copyin(a[0:N-1][0:N-1]) \  
        copyout(maxchange)
```

```
    {  
    for (int iter=0; iter<niters; iter++) {  
        maxchange = 0.;  
        {  
        #pragma acc region  
        for (int i=1; i<N-1; ++i)  
            for (int j=1; j<N-1; ++j) {  
                tmp[j][i] = w0*a[j][i] +  
                    w1 * (a[j-1][i] + a[j][i-1] +  
                        a[j+1][i] + a[j][i+1]) +  
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +  
                        a[j+1][i+1] + a[j-1][i+1]);  
                double change = fabs(tmp[i][j] - a[i][j]);  
                maxchange = fmax(maxchange, change);  
            }  
        }  
    }  
    #pragma acc region  
    for (int i=0; i<N-1; ++i)  
        for (int j=0; j<N-1; ++j)  
            a[j][i] = tmp[j][i];  
    }  
    }  
    }  
    twodFree(tmp);  
    return maxchange;  
}
```

Note funny loop ordering



```
twodFree(tmp);  
return maxchange;
```

```
double jacobi(double **restrict a, const int N, const int niters) {
```

```
    const double w0 = -15./6., w1 = 4./3., w2 = -1./12;
```

```
    double **restrict tmp = twodMalloc(N,N);
```

```
    double maxchange;
```

```
    #pragma acc data region copyout(tmp[0:N-1][0:N-1]) \
        copyin(a[0:N-1][0:N-1]) \
        copyout(maxchange)
```

```
    {
    for (int iter=0; iter<niters; iter++) {
        maxchange = 0.;
        {
        #pragma acc region
        for (int i=1; i<N-1; ++i)
            for (int j=1; j<N-1; ++j) {
                tmp[j][i] = w0*a[j][i] +
                    w1 * (a[j-1][i] + a[j][i-1] +
                        a[j+1][i] + a[j][i+1]) +
                    w2 * (a[j-1][i-1] + a[j+1][i-1] +
                        a[j+1][i+1] + a[j-1][i+1]);
                double change = fabs(tmp[i][j] - a[i][j]);
                maxchange = fmax(maxchange, change);
            }
        }
    }
    #pragma acc region
    for (int i=0; i<N-1; ++i)
        for (int j=0; j<N-1; ++j)
            a[j][i] = tmp[j][i];
    }
    }
}
```

Multid loops must  
be rectangular



```
twodFree(tmp);
return maxchange;
```

```
}
```

# Cannot appear in compute region:

- jumps out of region
- non-inlinable function calls

# Compiler feedback

- non-stride-1 access (uncoalesced)
- loop schedule (can be overridden)
- inferred data movement
- lack of parallelization due to - required  
privitization, data dependency
  - Not enough feedback here!



# accelmath

- `#include <accelmath.h>`
- fast GPU math library (exp, sin, cos...)

# accel.h

- `#include <accel.h>`
- Gives you access to  
`acc_get_num_devices()`,  
`acc_set_device_num()`
- Can also set by environment variable
- Each thread/MPI task can set own device

# OpenACC

- Version 1.0 this summer (12.8)
- Early betas now available
- OpenACC-inspired features now part of a technical report proposed for OpenMP 4.0
- Still need to understand what's going on under the hood (CUDA) to make code fast