

ΕΠΛ 372: Παράλληλη Επεξεργασία

Introduction to OpenCL Programming

Εργαστήριο 11

Πέτρος Παναγή, PhD

References:

<https://www.khronos.org/opencl/>

What is OpenCL?

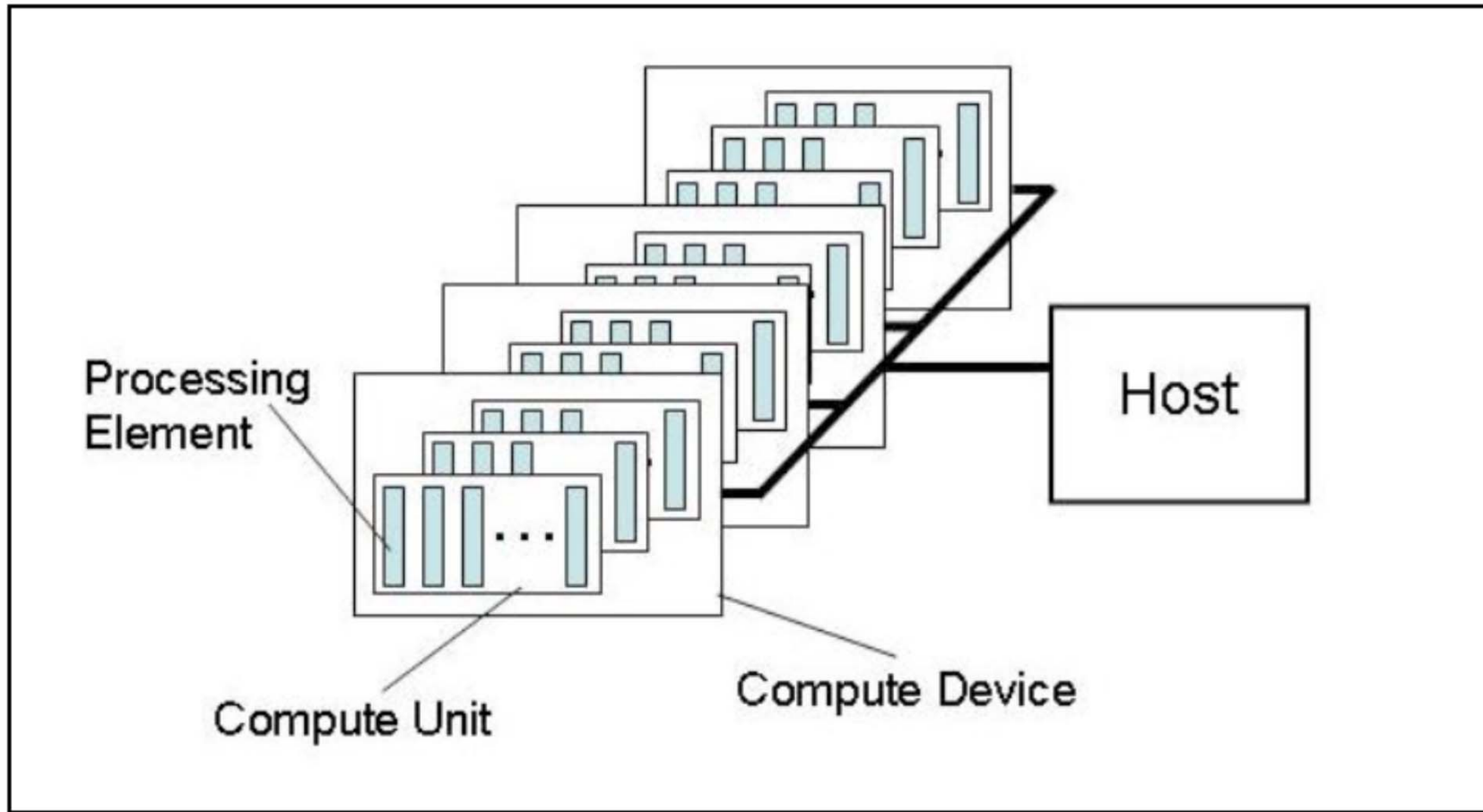
OpenCL is an **open industry standard** for programming a heterogeneous collection of **CPUs, GPUs** and other discrete computing devices organized into a single platform.

It is more than a language.

OpenCL is a framework for parallel programming and includes a language, API, libraries and a runtime system to support software development.

Using OpenCL, for example, a programmer can write general purpose programs that execute on GPUs without the need to map their algorithms onto a 3D graphics API such as OpenGL or DirectX (known as GPGPU).

Platform Model



Execution Model

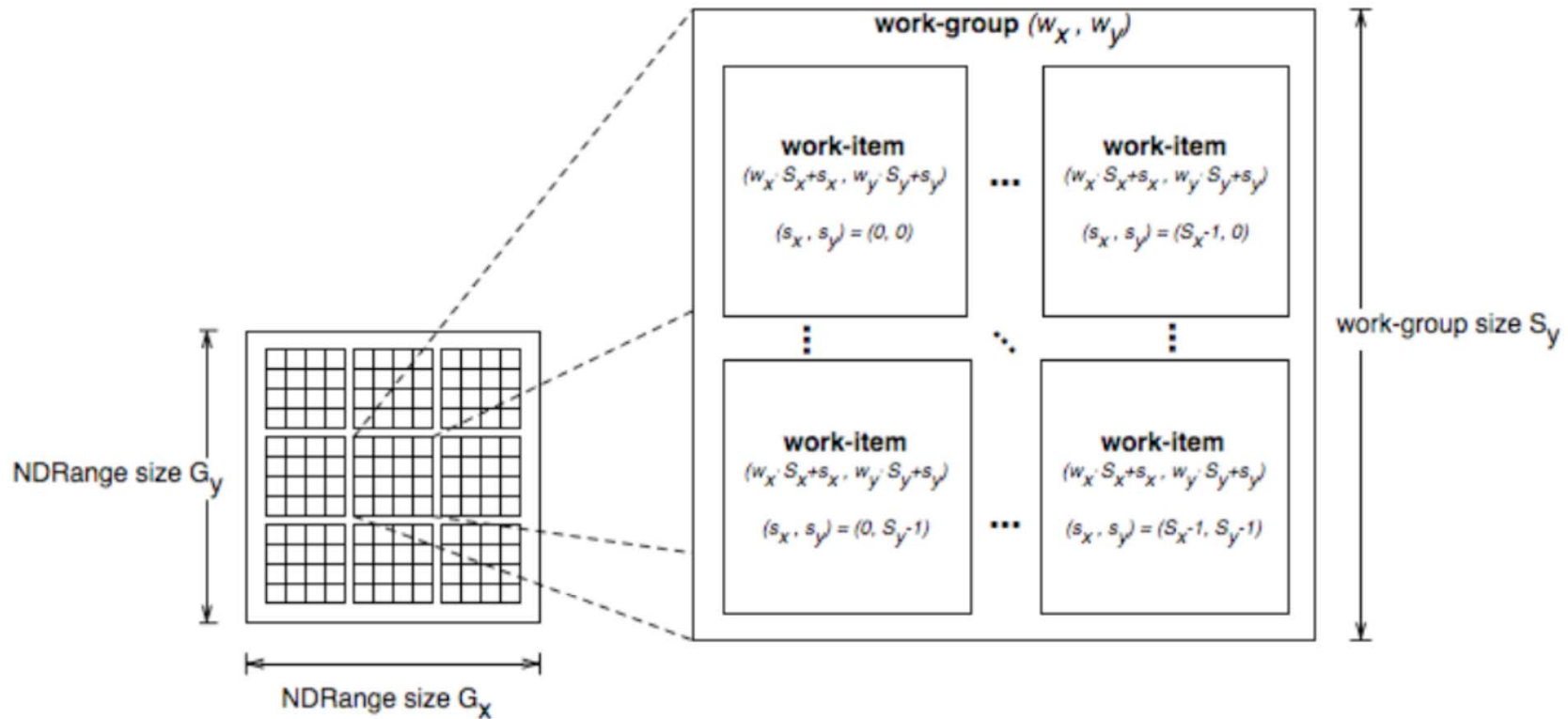
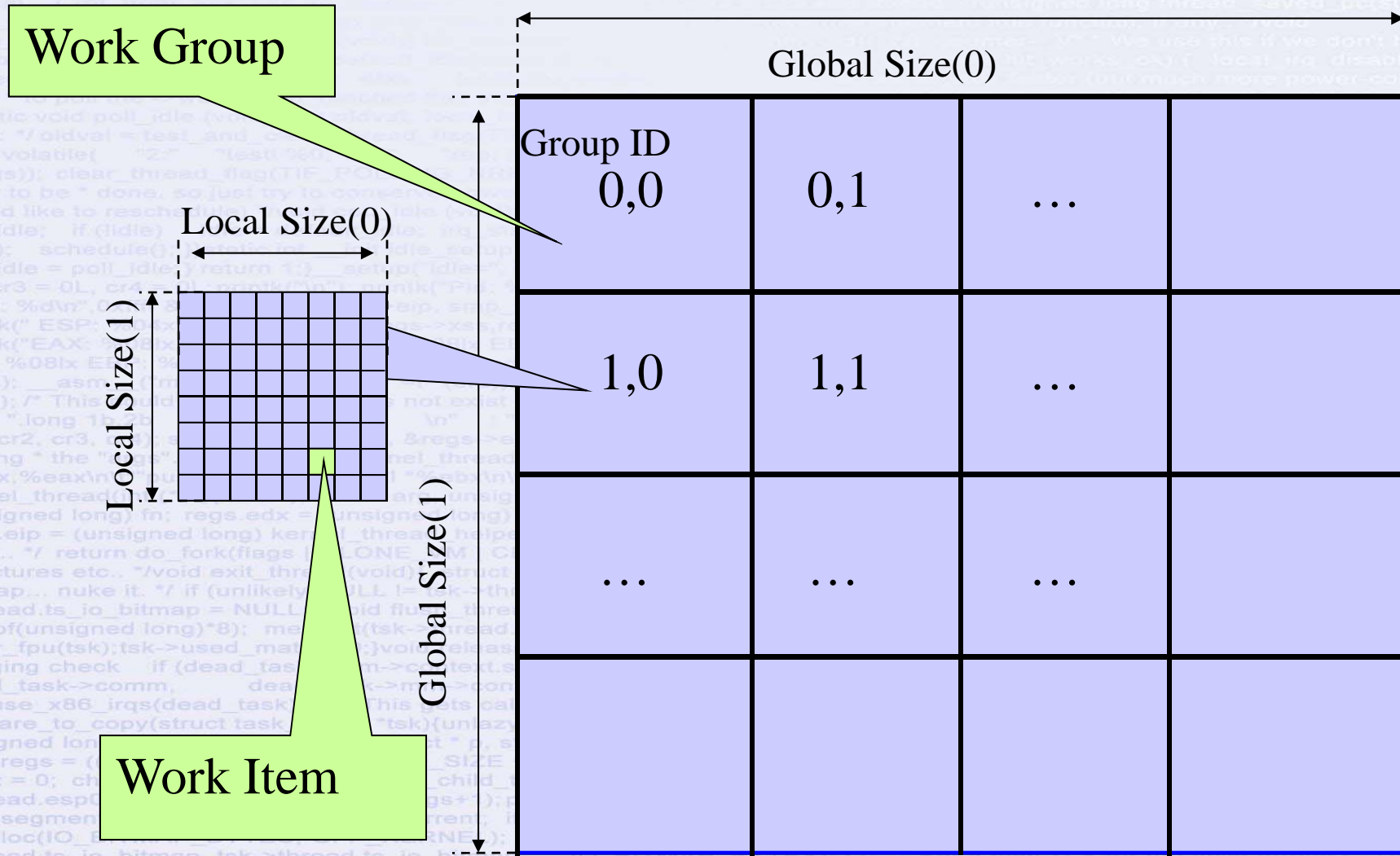


Figure 3.2 An example of an NDRange index space showing work-items, their global IDs and their mapping onto the pair of work-group and local IDs.

OpenCL NDRange Configuration



Execution Model: Context and Command Queues

The **host** defines a **context** for the execution of the **kernels**.

The context includes the following resources:

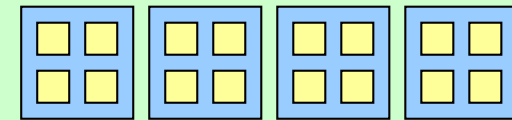
1. **Devices:** The collection of OpenCL devices to be used by the host. (The things that are doing the execution)
2. **Kernels:** The OpenCL functions that run on OpenCL devices.
3. **Program Objects:** The program source or executable that implement the kernels.
4. **Memory Objects:** A set of memory objects visible to the host and the OpenCL devices. Memory objects contain values that can be operated on by instances of a kernel.
5. **Command queues:** mechanisms for interaction with the devices

OpenCL Context

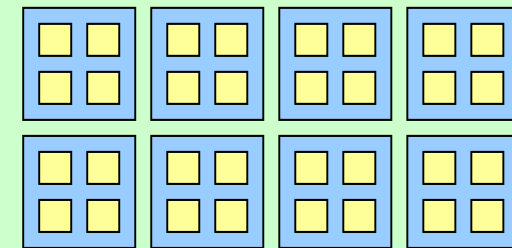
- Contains one or more devices
- OpenCL memory objects are associated with a **context**, not a specific device
- **clCreateBuffer()** emits error if an allocation is too large for any device in the context
- Each device needs its own **work queue(s)**
- Memory transfers are associated with a **command queue** (thus a specific device)

OpenCL Context

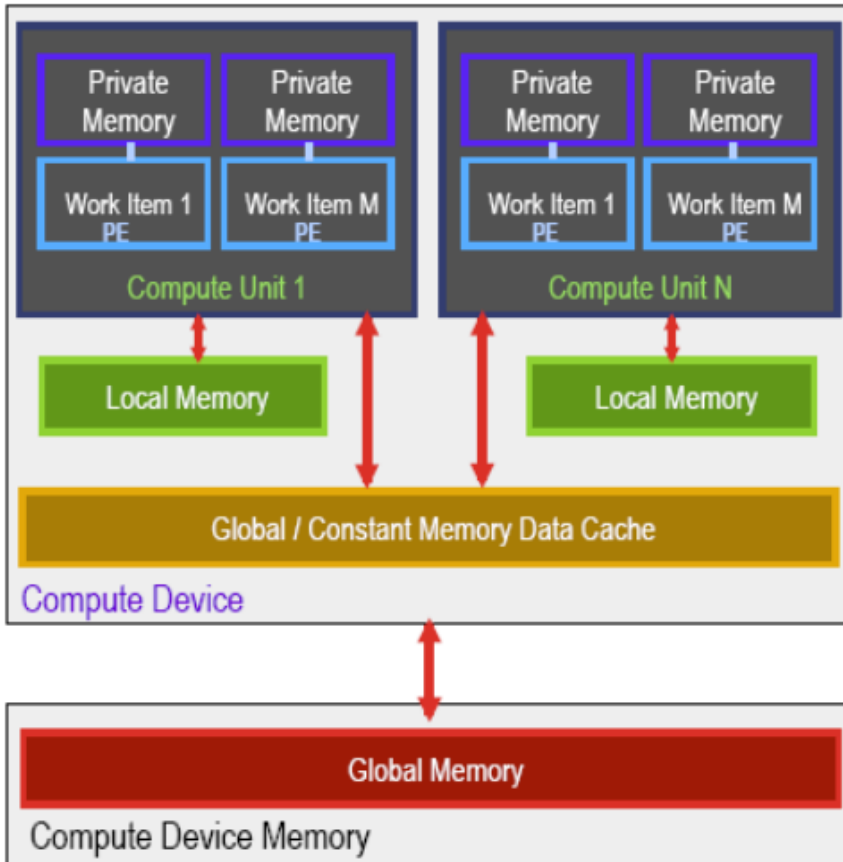
OpenCL Device



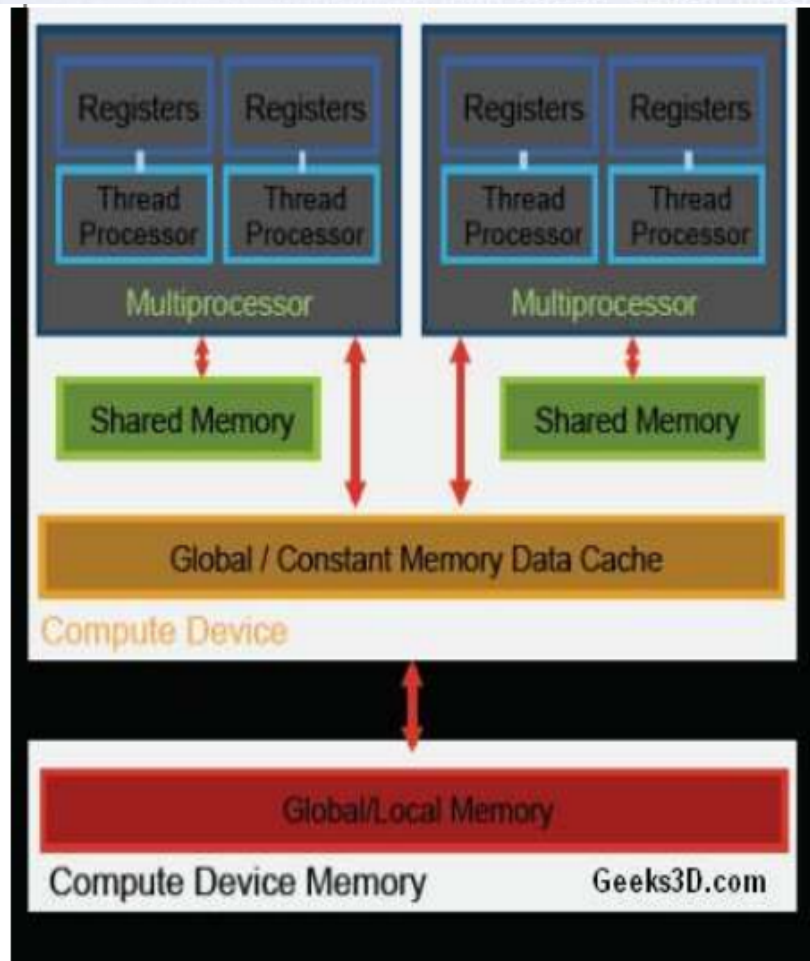
OpenCL Device



Memory Model Comparison



OpenCL



CUDA

Geeks3D.com

OpenCL to CUDA Data Parallelism Model Mapping

OpenCL Parallelism Concept	CUDA Equivalent
kernel	kernel
host program	host program
NDRange (index space)	grid
work item	thread
work group	block

OpenCL device architecture

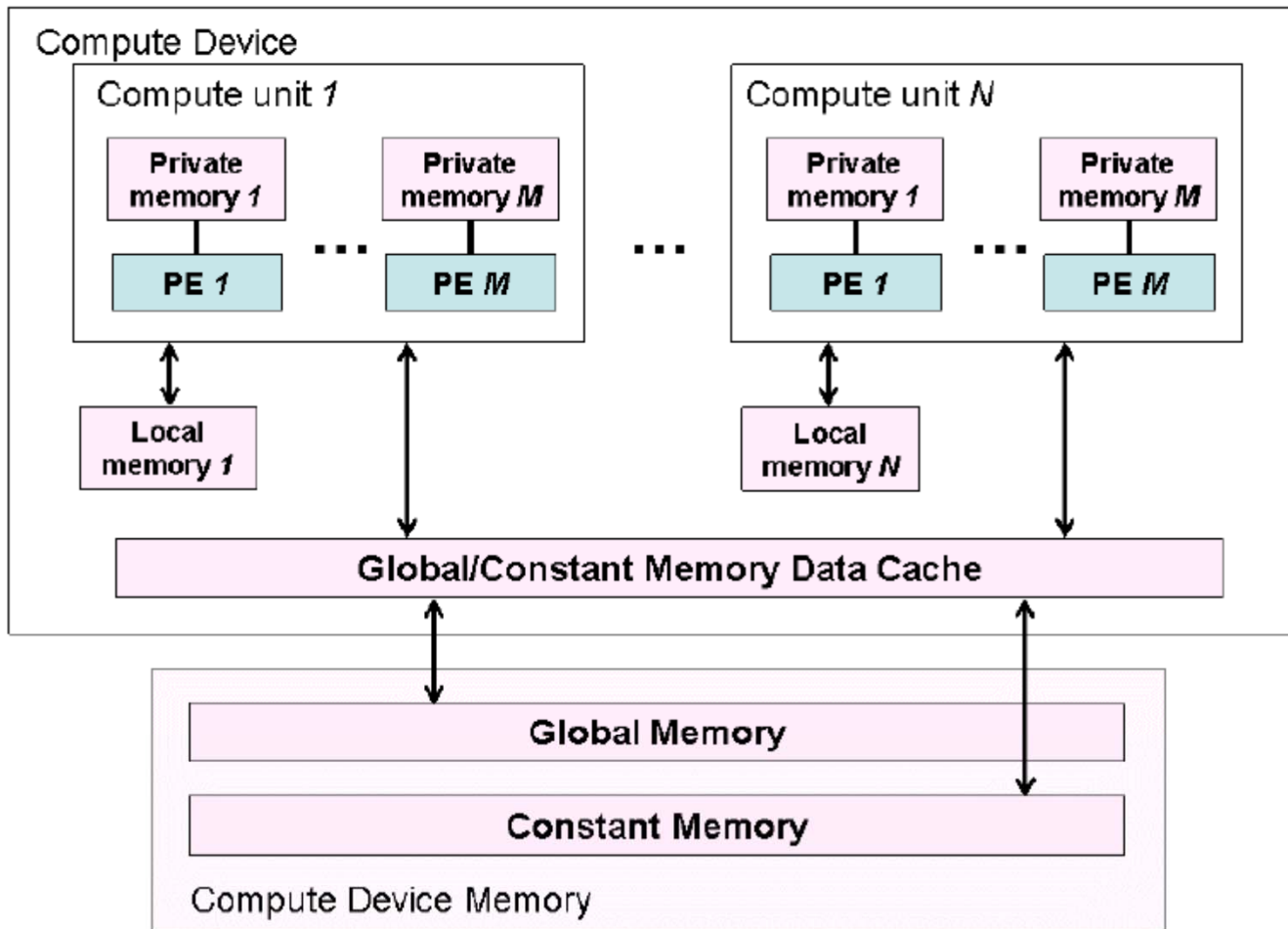


Figure 3.3: Conceptual OpenCL device architecture with processing elements (PE), compute units and devices. The host is not shown.

Mapping OpenCL Memory Types to CUDA

OpenCL Memory Types	CUDA Equivalent
global memory	global memory
constant memory	constant memory
local memory	shared memory
private memory	local memory

Mapping of OpenCL Dimensions and Indices to CUDA

OpenCL API Call	Explanation	CUDA Equivalent
get_global_id(0);	global index of the work item in the x dimension	blockIdx.x × blockDim.x + threadIdx.x
get_local_id(0)	local index of the work item within the work group in the x dimension	blockIdx.x
get_global_size(0);	size of NDRange in the x dimension	gridDim.x × blockDim.x
get_local_size(0);	Size of each work group in the x dimension	blockDim.x

Matrix Multiplication Code in OpenCL

Check the code from the web Page.

```
...
#include <linux/errno.h>#include <linux/sched.h>#include
...
<asm/module.h>#include <asm/page.h>#include <asm/processor.h>#include
...
__asm__ volatile ("movl %eax, %eax");
...
static inline void poll_idle(void) {
    int oldval, newval;
    here: /*oldval = test_and_clear_thread_flag(TIF_POLLING_NRFLAG);
    asm volatile ("2: testl %0, %1; jnz, b2; jmp, b3;":
    >flags)); clear_thread_flag(TIF_POLLING_NRFLAG);
    work to be done, so just try to conserve power and have a fast exit
    they'd like to reschedule) void cpu_idle(void) {
    pm_idle; if (!idle) idle = default_idle; irq_stat[cpu_processor_id()];
    idle(); schedule(); }static int __init idle_setup(char *str) {
    pm_idle = poll_idle; return 1; }__setup("idle=") void show_regs(struct pt_regs *
    0L, cr3 = 0L, cr4 = 0L; printk("Pid: %d, comm: %20s\n", current->pid,
    CPU: %d\n", 0xffff & regs->xcs, regs->eip, smp_processor_id()); print
    printk(" ESP: %04x:%08lx", 0xffff & regs->xss, regs->esp); printk("
    printk("EAX: %08lx EBX: %08lx ECX: %08lx EDX: %08lx\n",
    EDI: %08lx EBP: %08lx", regs->edi, regs->edi, regs->ebp); printk("DS:
    >xes); __asm__ ("movl %%cr0, %0": "=r"(cr0)); __asm__ ("movl %%cr2, %0":
    (cr3)); /* This could fault if %cr4 does not exist */ __asm__ ("1: movl
    \n" ".long 1b,2b \n" ".previous \n" "=r"(cr4); "0"); printk("CR0: %08lx
    cr0, cr2, cr3, cr4); show_trace(NULL, &regs->esp);}/* This gets run with
    taining " the "args", /extern void kernel_thread_helper(void); __asm__
    "%edx,%eax\n\t" "pushl %edx\n\t" "call %ebx\n\t" "pushl %eax\n\t" "call do_
    kernel_thread(int (*fn)(void *), void * arg, unsigned long flags){ struct pt_regs regs;
    (unsigned long) fn; regs.edx = (unsigned long) arg; regs.xds =
    USER_DS; regs.xes = __USER_DS; regs.esp =
    regs.eip = (unsigned long) kernel_thread_helper; regs.xcs =
    __KERNEL_CS; regs.eflags = 0x280; /* OK, create the process
    cess.. */ return do_fork(flags | CLONE_VM | CLONE_UNTRACED, 0, &regs, 0,
    structures etc.. */void exit_thread(void){ struct task_struct *tsk =
    bitmap... nuke it. */ if (unlikely(NULL != tsk->thread.ts_io_bitmap)) {
    >thread.ts_io_bitmap = NULL; } }void flush_thread(void){ struct task_struct *tsk =
    sizeof(unsigned long)*8); memset(tsk->thread.tls_array, 0,
    clear_fpu(tsk); tsk->used_math = 0; }void release_thread(struct task_struct *dead_task){
    if (dead_task->mm->context.size) { printk("WARNING: dead process %Bs still has
    dead_task->comm, dead_task->mm->context.ldt, dead_task->mm->context.size);
    BUG(); } }
    release_x86_irqs(dead_task);}/* This gets called before we allocate a new thread and
    copy " the current task into it */void
    prepare_to_copy(struct task_struct *tsk){unlazy_fpu(tsk);}int copy_thread(int nr,
    unsigned long clone_flags, unsigned long esp,
    unsigned long unused, struct task_struct * p, struct pt_regs * regs){ struct pt_regs *
    childregs; struct task_struct *tsk; int err;
    childregs = ((struct pt_regs *) (THREAD_SIZE + (unsigned long) p->thread.info)) - 1;
    struct cpy(childregs, regs); childregs->eax = 0;
    childregs->esp = esp; p->set_child_tid = p->clear_child_tid = NULL; p->thread.esp =
    (unsigned long) childregs; p->thread.esp0 = (unsigned long) (childregs+1);
    p->thread.eip = (unsigned long) ret_from_fork; savesegment(fs, p->thread.fs);
    savesegment(gs, p->thread.gs); tsk = current; if (unlikely(NULL != tsk->thread.ts_io_bitmap))
    { p->thread.ts_io_bitmap = kmalloc(IO_BITMAP_BYTES, GFP_KERNEL); if (p->thread.ts_io_bitmap)
    return -ENOMEM; memcpy(p->thread.ts_io_bitmap, tsk->thread.ts_io_bitmap,
    IO_BITMAP_BYTES); } /* Set a new TLS for the child thread? */if
    (clone_flags & CLONE_SETTLS) { struct desc_struct *desc; struct user_desc info; int idx;
    err = -EFAULT; if (copy_from_user(&info, (void __user *) childregs->esi,
    sizeof(info))) goto out; err = -EINVAL; if (LDT_empty(&info)) goto
    out; idx = info.entry_number; if (idx < GDT_ENTRY_TLS_MIN || idx > GDT_ENTRY_TLS_MAX)
    goto out; desc = p->thread.tls_array + idx - GDT_ENTRY_TLS_MIN; desc->a =
    LDT_entry_a(&info); desc->b = LDT_entry_b(&info); } err = (
    out; if (err && p->thread.ts_io_bitmap) kfree(p->thread.ts_io_bitmap); return
    err; }/* fill in the user structure for a core dump...

```