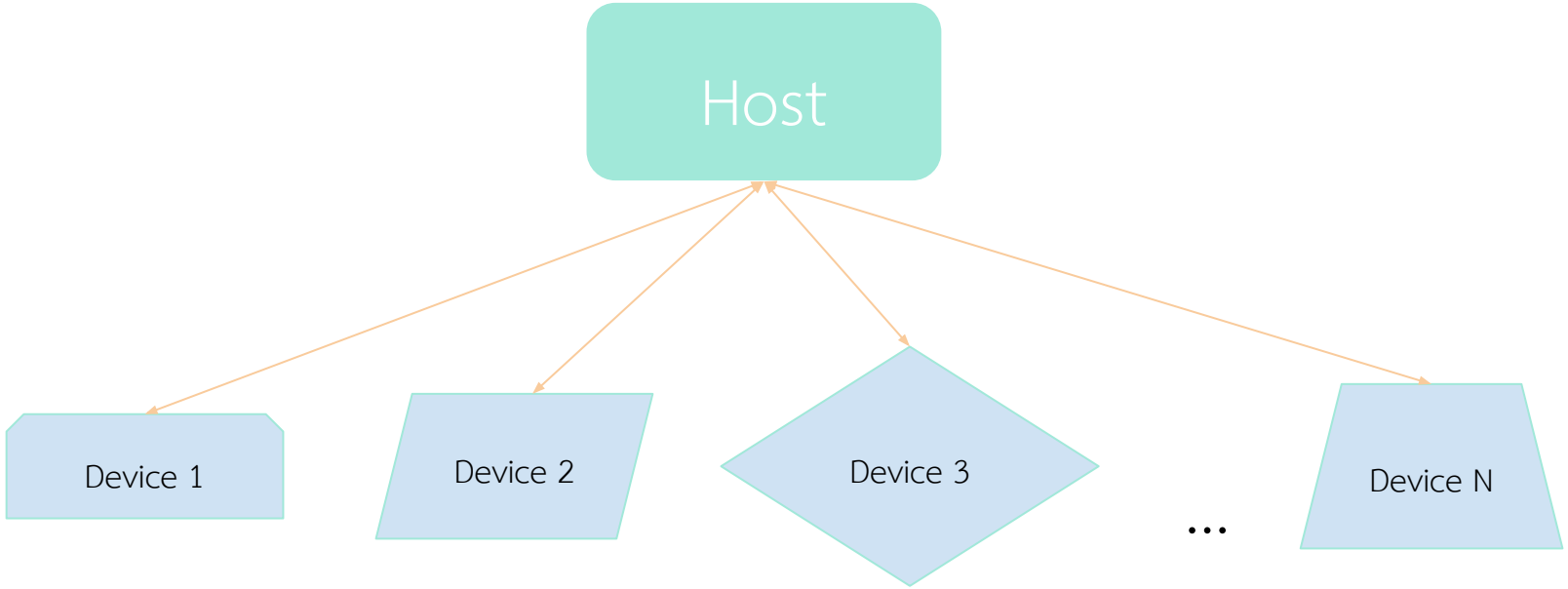
OpenCL Programming

Pisit Makpaisit

OpenCL

- Framework สำหรับ Heterogenous Computing (CPU, GPU, FPGA, DSP และ Accelerator อื่นๆ)
  - แต่ละหน่วยประมวลผลมีความสามารถที่ไม่เหมือนกัน แต่ต้องการโปรแกรมที่สามารถใช้งานหน่วยประมวลผลทั้งหมดพร้อมกันได้
- เป็น Library based
- ใช้ได้กับหลากหลาย device และไม่ขึ้นกับ vendor
- เขียนโปรแกรมเพียงครั้งเดียวและใช้ได้กับหลาย device
- สามารถเขียนบน Android ได้
  - https://software.intel.com/en-us/android/articles/opencl-basic-sample-for-android-os

OpenCL Concept



Terminology

- **Device** - compute device มีหลายประเภท เช่น *CL_DEVICE_TYPE_CPU*, *CL_DEVICE_TYPE_GPU*, *CL_DEVICE_TYPE_ACCELERATOR* (accelerator อื่นๆ ที่คุยกับ host ผ่าน PCI)
- **Kernel** - ชุดของคำสั่งที่ประมวลที่ประมวลผลบน device / หน่วยการทำงานบน device
- **Context** - ประกอบด้วย
  - เซ็ตของ device
  - หน่วยความจำ
  - command queues

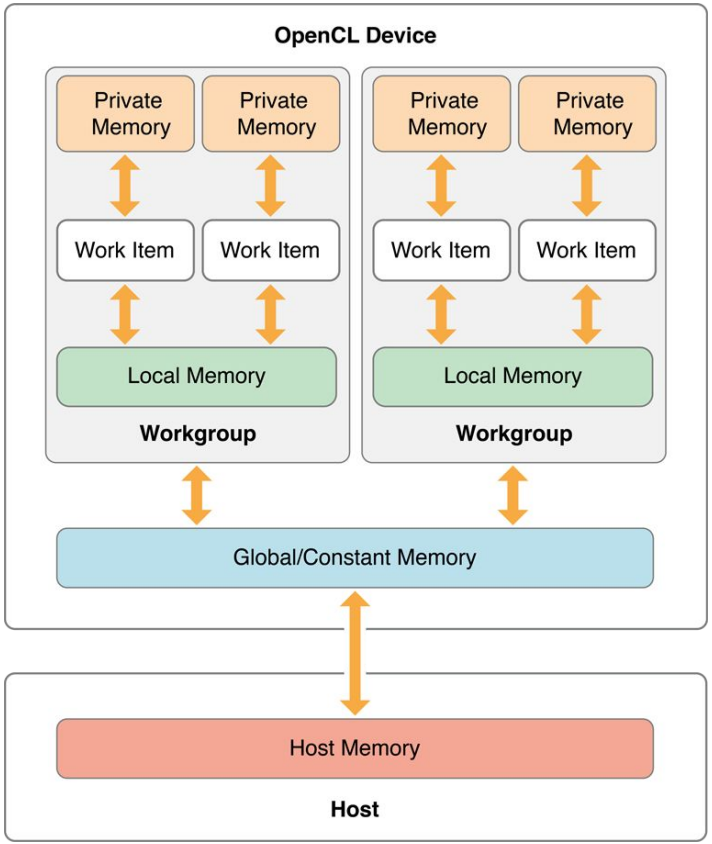
Terminology

- **Command-queue** - คิวของคำสั่งที่ส่งไปยัง device
  - command-queue 1 อันจะเชื่อมไปยัง device เดียว แต่ device นึงสามารถมีได้หลาย command-queue
  - จัดการรูปแบบลำดับของการทำงาน (ก่อนหลัง / พร้อมกัน)
- **Platform** - host + หลาย devices
- **Work item** - หน่วยย่อยสำหรับประมวลผลบน device
- **Work group** - กลุ่มของ work item

OpenCL Memory Model

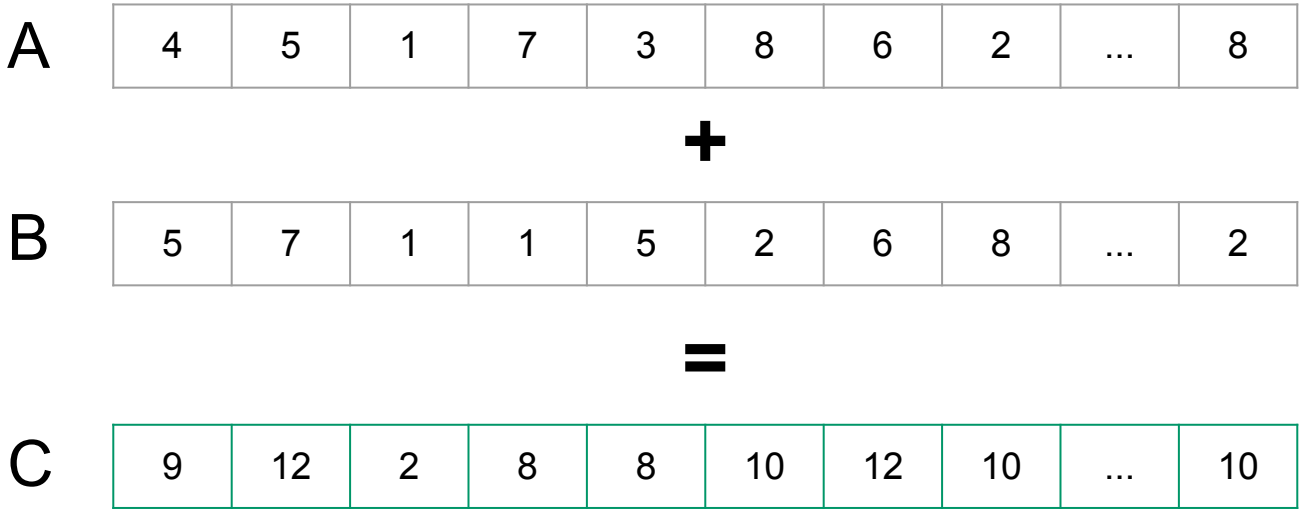
- Private memory - ใช้ได้เฉพาะจากใน work item เดียวกัน
- Local memory - แชร์กันระหว่างใน work group เดียวกัน
- Global memory - ใช้ได้จากทุก work item และสามารถอ่านและเขียนจาก host ได้
- Constant memory - เหมือน global memory แต่อ่านได้อย่างเดียว

* ต้องย้ายข้อมูลระหว่าง host และ device ด้วยตัวเอง



https://developer.apple.com/library/content/documentation/Performance/Conceptual/OpenCL_MacProgGuide/OpenCLLionMemoryObjects/OpenCLLionMemoryObjects.html

Vector Addition



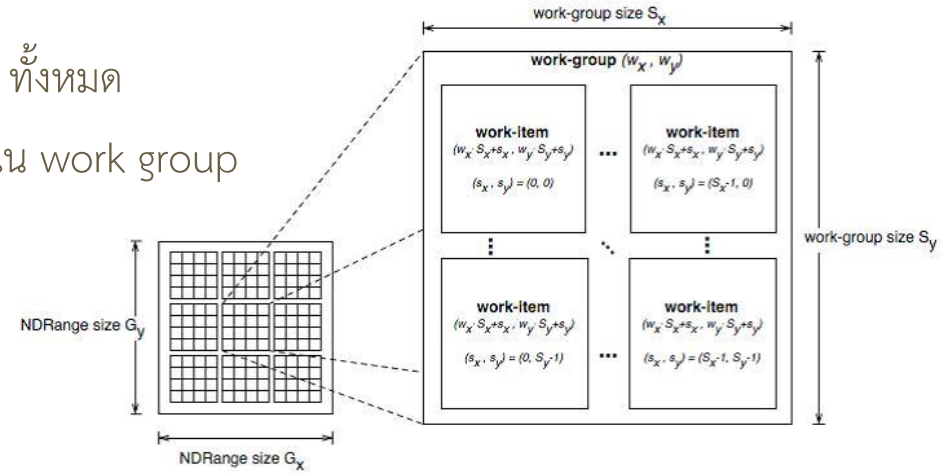
Kernels

```
__kernel void vecadd(__global const double* a,
                     __global const double* b,
                     __global double* c,
                     const int n)
{
  int gid = get_global_id(0);

  if (gid < n)
    c[gid] = a[gid] + b[gid];
}
```

Execution Model

- Global Dimensions - ระบุจำนวน work item ทั้งหมด
- Local Dimensions - ระบุจำนวน work item ใน work group



- work item ใน work group เดียวกัน
  - สามารถใช้ local memory ร่วมกัน
  - synchronize เฉพาะใน group ได้
- การกำหนดจำนวน global/local dimension มีผลต่อความเร็ว ต้องเลือกให้เหมาะสมกับอัลกอริทึมที่ใช้

https://software.intel.com/sites/landingpage/opencl/optimization-guide/Basic_Concepts.htm

Vector Addition Program

1. ดึงข้อมูล platform_id

```
clGetPlatformIDs(1, &platform_id, NULL);
```

2. ดึงข้อมูล device_id

```
clGetDeviceIDs(platform_id, CL_DEVICE_TYPE_GPU, 1, &device_id, NULL);
```

3. สร้าง context จาก device_id ที่กำหนด (ให้ context มี device แค่ตัวเดียว)

```
context = clCreateContext(0, 1, &device_id, 0, 0, &cl_ret);
```

4. สร้าง command queue สำหรับส่งคำสั่งไปยัง device

```
queue = clCreateCommandQueue(context, device_id, 0, &cl_ret);
```

Vector Addition Program

5. จองหน่วยความจำบน device

```
d_a = clCreateBuffer(context, CL_MEM_READ_ONLY, n*sizeof(double), NULL, &cl_ret); d_b
= clCreateBuffer(context, CL_MEM_READ_ONLY, n*sizeof(double), NULL, &cl_ret);
d_c = clCreateBuffer(context, CL_MEM_WRITE_ONLY, n*sizeof(double), NULL, &cl_ret);
```

Building Kernel Program

สร้างจากโค้ด kernel ในไฟล์ .cl

```
vecadd_kernel_file = fopen("vecadd.cl", "r");
kernel_source = (char*) malloc(MAX_KERNEL_SIZE * sizeof(char));
fread(kernel_source, sizeof(char), MAX_KERNEL_SIZE * sizeof(char),
                vecadd_kernel_file);

program = clCreateProgramWithSource(context, 1, (const char **)&kernel_source,
                NULL, &cl_ret);
cl_ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
vecadd_kernel = clCreateKernel(program, "vecadd", &cl_ret);
```

หรือสร้างจากสตริงในโปรแกรม

```
const char *kernel_source =                                     "\n" \

"__kernel void vecadd(__global const double* a,                 \n" \
"                     __global const double* b,                 \n" \
"                     __global double* c,                       \n" \
"                     const int n)                              \n" \
"{                                                              \n" \
"  int gid = get_global_id(0);                                  \n" \
"                                                               \n" \
"  if (gid < n)                                                 \n" \
"    c[gid] = a[gid] + b[gid];                                  \n" \
"}                                                              \n";
program = clCreateProgramWithSource(context, 1, (const char **)&kernel_source,
              NULL, &cl_ret);
cl_ret = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
vecadd_kernel = clCreateKernel(program, "vecadd", &cl_ret);
```

Vector Addition Program

7. ระบุว่าจะใช้ตัวแปรใดเป็น argument ของ kernel

```
cl_ret = clSetKernelArg(vecadd_kernel, 0, sizeof(cl_mem), (void*)&d_a);
cl_ret = clSetKernelArg(vecadd_kernel, 1, sizeof(cl_mem), (void*)&d_b);
cl_ret = clSetKernelArg(vecadd_kernel, 2, sizeof(cl_mem), (void*)&d_c);
cl_ret = clSetKernelArg(vecadd_kernel, 3, sizeof(cl_int), (void*)&n);
```

8. ส่งข้อมูลจาก host -> device

```
cl_ret = clEnqueueWriteBuffer(queue, d_a, CL_TRUE, 0, sizeof(double) * n,
                a, 0, NULL, NULL);
cl_ret = clEnqueueWriteBuffer(queue, d_b, CL_TRUE, 0, sizeof(double) * n,
                b, 0, NULL, NULL);
```

9. execute kernel

```
local_dim = 64;
global_dim = ((n + local_dim - 1) / local_dim) * local_dim;
clEnqueueNDRangeKernel(queue, vecadd_kernel, 1, NULL,
                &global_dim, &local_dim, 0, NULL, NULL);
```

10. รอจนทุกคำสั่งใน command queue เสร็จ

```
clFinish(queue);
```

11. ส่งคำตอบกลับจาก device

```
cl_ret = clEnqueueReadBuffer(queue, d_c, CL_TRUE, 0, sizeof(double) * n,
                c, 0, NULL, NULL);
```

Compile OpenCL Source Code

เพิ่ม option “-lOpenCL” ตอนคอมไพล์โปรแกรม

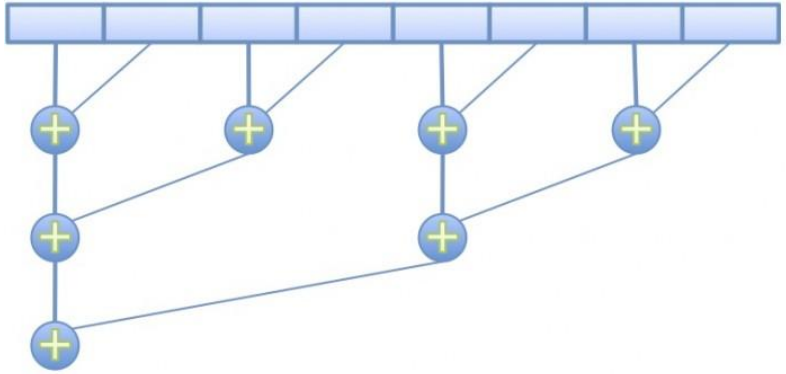
เช่น

```
gcc vecadd.c -o vecadd -lOpenCL
```

Parallel Reduction

รับข้อมูลเป็นเซ็ตของตัวเลข และคืนข้อมูลกลับไปเป็นค่าหนึ่ง เช่น

- Summation
- Maximum
- Minimum



Parallel Reduction Tree for Associative Operator

SIMD Utilization for Reduction Tree

```
__kernel void summation(__global const double *array,
                        __global double *global_sums,
                        __local double *local_sums,
                        const int n)
{
  int gid = get_global_id(0), lid = get_local_id(0), group_size = get_local_size(0);

  if (gid < n)  local_sums[lid] = array[gid];
  else          local_sums[lid] = 0;

  for (int stride = group_size / 2; stride > 0; stride /= 2) {
    if (lid < stride) local_sums[lid] += local_sums[lid + stride];
    barrier(CLK_LOCAL_MEM_FENCE);
  }

  if (lid == 0) global_sums[gid] = local_sums[0];
}
```

Exercise

1. ทดลอง implement parallel reduction
2. Vector dot product

Reference

- OpenCL 2.0 Specification - https://www.khronos.org/registry/OpenCL/specs/opencl-2.0.pdf
- OpenCL Optimization Case Study: Simple Reductions - http://developer.amd.com/resources/articles-whitepapers/opencl-optimization-case-study-simple-reductions/
- OpenCL An Introduction for HPC programmers, Tim Mattson, Intel