ผนวก ก

ตัวอย่างโปรแกรม

การคูณเมตริกซ์แบบ Sequential

```
void
matrixMul(float* C, const float* A, const float* B, unsigned int hA, unsigned int wA,
unsigned int wB)
{
    for (unsigned int i = 0; i < hA; ++i)
        for (unsigned int j = 0; j < wB; ++j) {
            float sum = 0;
            for (unsigned int k = 0; k < wA; ++k) {
                float a = A[i * wA + k];
                float b = B[k * wB + j];
                sum += a * b;
            }
            C[i * wB + j] = (float)sum;
        }
}
```

การคูณเมตริกซ์แบบ Parallel

การคูณเมตริกซ์แบบ Parallel นี้เป็นตัวอย่างการคูณโดยใช้ CUDA

```
__global__ void
matrixMul( float* C, float* A, float* B, int wA, int wB)
{
    // Block index
    int bx = blockIdx.x;
    int by = blockIdx.y;
```

```
// Thread index
int tx = threadIdx.x;
int ty = threadIdx.y;

// Index of the first sub-matrix of A processed by the block
int aBegin = wA * BLOCK_SIZE * by;

// Index of the last sub-matrix of A processed by the block
int aEnd   = aBegin + wA - 1;

// Step size used to iterate through the sub-matrices of A
int aStep  = BLOCK_SIZE;

// Index of the first sub-matrix of B processed by the block
int bBegin = BLOCK_SIZE * bx;

// Step size used to iterate through the sub-matrices of B
int bStep  = BLOCK_SIZE * wB;

// Csub is used to store the element of the block sub-matrix
// that is computed by the thread
float Csub = 0;

// Loop over all the sub-matrices of A and B
// required to compute the block sub-matrix
for (int a = aBegin, b = bBegin;
        a <= aEnd;
        a += aStep, b += bStep) {
```

```
    // Declaration of the shared memory array As used to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each thread loads
    // one element of each matrix
    AS(ty, tx) = A[a + wA * ty + tx];
    BS(ty, tx) = B[b + wB * ty + tx];

    // Synchronize to make sure the matrices are loaded
    __syncthreads();

    // Multiply the two matrices together;
    // each thread computes one element
    // of the block sub-matrix
    for (int k = 0; k < BLOCK_SIZE; ++k)
        Csub += AS(ty, k) * BS(k, tx);

    // Synchronize to make sure that the preceding
    // computation is done before loading two new
    // sub-matrices of A and B in the next iteration
    __syncthreads();
}
```

```
    // Write the block sub-matrix to device memory;
    // each thread writes one element
    int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
    C[c + wB * ty + tx] = Csub;
}
```

ตัวอย่างการเข้าถึงและสั่งงานจีพียูด้วย CUDA Programming

การเข้าถึงและสั่งงานจีพียูด้วย CUDA Programming โดยปกติ สามารถสั่งงานจีพียูผ่านคำสั่งของคูด้าได้ 2 ลักษณะคือ CUDA runtime API และ CUDA driver API ดังตัวอย่าง ซึ่งการทำงานของแบ็คเอ็นด์ในระบบเวอร์ชวลคูด้าที่ได้พัฒนาขึ้นมานั้น จะทำงานโดยเข้าถึงจีพียูด้วยคำสั่งของคูด้าแบบ CUDA driver API โดยจะใช้ฟังก์ชั่นพื้นฐานที่ได้กล่าวไว้ในข้อ 3.1.1.2

ตัวอย่าง CUDA Programming แบบ CUDA runtime API

```
// Device code
__global__ void VecAdd(float* A, float* B, float* C)
{
int i = threadIdx.x;
if (i < N)
        C[i] = A[i] + B[i];
}


// Host code
int main()
{
// Allocate vectors in device memory
size_t size = N * sizeof(float);
float* d_A;
cudaMalloc((void**)&d_A, size);
float* d_B;
cudaMalloc((void**)&d_B, size);
float* d_C;
cudaMalloc((void**)&d_C, size);
// Copy vectors from host memory to device memory
// h_A and h_B are input vectors stored in host memory
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
// Invoke kernel
int threadsPerBlock = 256;
int blocksPerGrid = (N + threadsPerBlock – 1) / threadsPerBlock;
VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C);
// Copy result from device memory to host memory
// h_C contains the result in host memory
cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
// Free device memory
cudaFree(d_A);
cudaFree(d_B);
cudaFree(d_C);
}
```

ที่มา : “CUDA Programming Guide Version 2.3” โดย NVIDIA CUDA™ จาก http://www.nvidia.com

ตัวอย่าง CUDA Programming แบบ CUDA driver API

```
// Host code
int main()
{
// Initialize
```

```
if (cuInit(0) != CUDA_SUCCESS)
        exit (0);
// Get number of devices supporting CUDA
int deviceCount = 0;
cuDeviceGetCount(&deviceCount);
if (deviceCount == 0) {
        printf("There is no device supporting CUDA.\n");
        exit (0);
}
// Get handle for device 0
CUdevice cuDevice = 0;
cuDeviceGet(&cuDevice, 0);
// Create context
CUcontext cuContext;
cuCtxCreate(&cuContext, 0, cuDevice);
// Create module from binary file
CUmodule cuModule;
cuModuleLoad(&cuModule, “VecAdd.ptx”);
// Get function handle from module
CUfunction vecAdd;
cuModuleGetFunction(&vecAdd, cuModule, "VecAdd");
// Allocate vectors in device memory
size_t size = N * sizeof(float);
CUdeviceptr d_A;
cuMemAlloc(&d_A, size);
CUdeviceptr d_B;
cuMemAlloc(&d_B, size);
CUdeviceptr d_C;
cuMemAlloc(&d_C, size);
```

```
// Copy vectors from host memory to device memory
// h_A and h_B are input vectors stored in host memory
cuMemcpyHtoD(d_A, h_A, size);
cuMemcpyHtoD(d_B, h_B, size);
// Invoke kernel
#define ALIGN_UP(offset, alignment) \
        (offset) = ((offset) + (alignment) – 1) & ~((alignment) – 1)
int offset = 0;
void* ptr;
ptr = (void*)(size_t)d_A;
ALIGN_UP(offset, __alignof(ptr));
cuParamSetv(vecAdd, offset, &ptr, sizeof(ptr));
offset += sizeof(ptr);
ptr = (void*)(size_t)d_B;
ALIGN_UP(offset, __alignof(ptr));
cuParamSetv(vecAdd, offset, &ptr, sizeof(ptr));
offset += sizeof(ptr);
ptr = (void*)(size_t)d_C;
ALIGN_UP(offset, __alignof(ptr));
cuParamSetv(vecAdd, offset, &ptr, sizeof(ptr));
offset += sizeof(ptr);
cuParamSetSize(VecAdd, offset);
int threadsPerBlock = 256;
int blocksPerGrid =(N + threadsPerBlock – 1) / threadsPerBlock;
cuFuncSetBlockShape(vecAdd, threadsPerBlock, 1, 1);
cuLaunchGrid(VecAdd, blocksPerGrid, 1);
// Copy result from device memory to host memory
// h_C contains the result in host memory
cuMemcpyDtoH(h_C, d_C, size);
```

```
// Free device memory
cuMemFree(d_A); cuMemFree(d_B); cuMemFree(d_C);
}
```

ที่มา : “CUDA Programming Guide Version 2.3” โดย NVIDIA CUDA™ จาก http://www.nvidia.com