Cuda başlangıç

Include

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
```

Önemli birtakım komutlar

Bellek tahsisi

```
size_t bytes = N * N * sizeof(float);// N*N'lik float size'ı
float *data;
cudaMalloc(&data, bytes);` //bytes kadar yer açılıyor ve açılan yerin başlangıç adresini data
değişkenine atıyor d_a ya eriştiğimizde o bellek alanına erişiyoruz
```

Kopyalama

```
cudaMemcpy(d_a, h_a.data(), bytes, cudaMemcpyHostToDevice);
```

CPU'dan GPU'ya matrisleri kopyalama || h_a.data() h_a vektörünün ilk elemanının adresini(pointerı) döndürür, sonra o adresten bytes kadarlık(bu durumda tüm h_a vektörü oluyor) veriyi d_a(GPU)'ya kopyalıyor

Pro Tips: Kopyalama işlemleri ağır çalışır, bu nedenle hızın önemsiz olduğu program başlangıcı veya sonunda kopyalama işlemlerini yapmak büyük performans kazancı olur.

Fonksiyon türleri

global:

`__global__` (ya da `__device__`): Bu etiket, bir fonksiyonun hem GPU hem de CPU tarafından çağrılabilir olduğunu belirtir. Genellikle, bu tür fonksiyonlar, bir GPU kernel fonksiyonu olarak kullanılır.

device:

`__device__` : Bu etiket, bir fonksiyonun yalnızca bir CUDA cihazı (GPU) tarafından çağrılabilir olduğunu belirtir. Bu tür fonksiyonlar, GPU üzerinde çalışan paralel CUDA kernel'ları içinde kullanılır.

`__device__` etiketi fonksiyonlar gibi değişkenlerin önüne eklendiğinde o değişkenin GPU üzerinde çalışacağını belirtir. Normal veri tanımlamaları GPU fonksiyonları içinde kullanılamaz(hata verir).
Örneğin:

```
int CPUNumber = 3;
__device__ int GPUNumber = 3;
__global__ void GPUPrint(){) {
    printf("%d",GPUNumber); (Doğru kullanım)
    printf("%d",CPUNumber); (Hata verir)
}
```

host:

`__host__` : Bu etiket, bir fonksiyonun yalnızca ana bilgisayar (host) tarafından çağrılabilir olduğunu belirtir. Bu tür fonksiyonlar, CPU üzerinde çalışan ana bilgisayar programı tarafından çağrılır.

Garbage Tip: `__host__ __device__` gibi bir tanımlama yapılınca hem GPU hem CPU üzerinde çalışıyor.

!!!!
device functions can be called only from the device, and it is executed only in the device.

global functions can be called from the host, and it is executed in the device.

Yukarıda yazdığı gibi global fonksiyonlar host(CPU) üzerinden çağrılabilir ancak device(GPU) üzerinde çalışır. Device ise sadece device içinden çağrılır ve çalışır.

Önerim 1 adet main global fonksiyonu yazıp sadece onu host içinde çağırmak, diğer GPU'da çalışacak kodları device ile tanımlayıp globalde çağırmak.

Pro Tips: Global ve device fonksiyonları içinde değişken tanımlarken `__device__` etiketine gerek yok çünkü zaten global veya device fonksiyonu içinde olduğumuzdan oluşturduğumuz değişkenlerde GPU değişkenleri oluyor otomatik olarak.(veri tanımlarken sadece device kullanılır, global sadece kernel fonksiyonlarda kullanılır)

Thread ve block

Thread:

- CUDA programlamasında, thread, GPU üzerinde bağımsız bir iş parçacığı olarak çalışan en küçük birimdir.
- Her bir thread, bir kernel fonksiyonunun bir örneği olarak düşünülebilir ve bu thread'ler paralel olarak çalışabilir.
- Thread'ler, belirli bir blok içinde veya birden çok blok arasında örgülenirler.
- Thread'ler, genellikle thread indeksi veya 3D uzayda belirli bir konumu temsil eden `threadIdx` yapısı ile tanımlanır.
  Block:
- Block, bir veya daha fazla thread içeren bir grup olarak düşünülebilir.
- Bloklar, GPU üzerinde bağımsız olarak çalışan ve kendi içinde iletişim kuran thread'leri gruplandırır.
- Blok sayısı ve her bir bloktaki thread sayısı, kernel fonksiyonunu başlatırken belirlenebilir.
- Thread'ler arasında paylaşılan bellek, aynı bloktaki thread'ler arasında erişilebilir.
  Not: Block'lar, birbirleriyle doğrudan iletişim kuramazlar. Her blok, kendi içindeki thread'ler arasında iletişim kurabilir, ancak farklı bloklar arasında iletişim kurma yetenekleri doğrudan sınırlıdır. Bloklar arasındaki iletişim genellikle daha üst seviyeli bir yönetim ile sağlanır. Örneğin, bloklar arası iletişim, paylaşılan bellek veya global bellek üzerinden gerçekleşebilir.

Grid:

- Grid, birden çok bloğun bir araya gelmesiyle oluşan bir yapıdır.
- Grid, tüm kernel fonksiyonu içindeki blokların toplamını temsil eder.
- Grid yapısı, kernel fonksiyonu başlatılırken belirlenen blok sayısına göre oluşur.

Block - thread tanımlama, fonksiyon çağırma ve 3D yapı

CUDA'nın dim3 parametresi ile block ve thread sayısını belirleyebiliyoruz.
Örneğin `dim3 gridSize(3,3,1)` ve `dim3 blockSize(3,3,1)` şeklinde 2 boyutlu toplam 9 block ve her block'ta 9 thread oluşturup:
`cudaFunc << < blockSize, threadSize >> > (x,y,z)` şeklinde cudaFunc adlı fonksiyona parametreleri(x,y,z) ile gönderiyoruz.

Not: gridSize ile kaç block olacağını, blockSize ile her blockta kaç thread olacağını söylemiş oluyoruz.

Yazılım için özet block ve thread kontrol kodları:

- `blockIdx.x` : Bu, hangi blokta olduğumuzu gösterir (0'dan başlar).
- `blockDim.x` : Bu, her blokta kaç thread olduğunu gösterir.
- `threadIdx.x` : Bu ise, o blokta hangi thread olduğumuzu gösterir.