CEN310 Paralel Programlama

Hafta-11 (İleri GPU Programlama)

Bahar Dönemi, 2024-2025

Genel Bakış

Konular

1. CUDA Bellek Modeli
2. Paylaşımlı Bellek Optimizasyonu
3. İş Parçacığı Senkronizasyonu
4. Performans Optimizasyon Teknikleri

Hedefler

- CUDA bellek hiyerarşisini anlamak
- Paylaşımlı bellek kullanımını öğrenmek
- İş parçacığı senkronizasyonunda ustalaşmak
- Optimizasyon stratejilerini uygulamak

Bellek Türleri

- Global Bellek
- Paylaşımlı Bellek
- Sabit Bellek
- Doku Belleği
- Yazmaçlar

Bellek Erişim Desenleri

```
// Birleştirilmiş bellek erişimi örneği
__global__ void birlesik_erisim(float* veri, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        // Birleştirilmiş erişim deseni
        float deger = veri[idx];
        // Değeri işle
        veri[idx] = deger * 2.0f;
    }
```

2. Paylaşımlı Bellek Optimizasyonu

Paylaşımlı Bellek Kullanımı

```
__global__ void matris_carpimi(float* A, float* B, float* C, int N) {
    __shared__ float paylasimliA[BLOK_BOYUTU][BLOK_BOYUTU];
    __shared__ float paylasimliB[BLOK_BOYUTU][BLOK_BOYUTU];

    int satir = blockIdx.y * blockDim.y + threadIdx.y;
    int sutun = blockIdx.x * blockDim.x + threadIdx.x;
    float toplam = 0.0f;

    for(int karo = 0; karo < N/BLOK_BOYUTU; karo++) {
        // Veriyi paylaşımlı belleğe yükle
        paylasimliA[threadIdx.y][threadIdx.x] =
            A[satir * N + karo * BLOK_BOYUTU + threadIdx.x];
        paylasimliB[threadIdx.y][threadIdx.x] =
            B[(karo * BLOK_BOYUTU + threadIdx.y) * N + sutun];

        __syncthreads();

        // Paylaşımlı bellek kullanarak hesapla
        for(int k = 0; k < BLOK_BOYUTU; k++) {
            toplam += paylasimliA[threadIdx.y][k] * paylasimliB[k][threadIdx.x];
        }

        __syncthreads();
    }

    C[satir * N + sutun] = toplam;
}
```

3. İş Parçacığı Senkronizasyonu

Senkronizasyon Yöntemleri

- Blok seviyesi senkronizasyon
- Izgara seviyesi senkronizasyon
- Atomik işlemler

Örnek: Atomik İşlemler

```
__global__ void histogram(int* veri, int* hist, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        atomicAdd(&hist[veri[idx]], 1);
    }
}
```

4. Performans Optimizasyonu

Optimizasyon Teknikleri

1. Bellek Birleştirme
2. Bank Çakışması Önleme
3. Doluluk Optimizasyonu
4. Döngü Açma

Örnek: Bank Çakışması Çözümü

```
// Kötü: Bank çakışmaları
__shared__ float paylasimli_veri[BLOK_BOYUTU][BLOK_BOYUTU];

// İyi: Bank çakışmalarını önlemek için dolgulu
__shared__ float paylasimli_veri[BLOK_BOYUTU][BLOK_BOYUTU + 1];
```

İleri Bellek Yönetimi

Birleşik Bellek

```
// Birleşik bellek ayır
float* birlesik_veri;
cudaMallocManaged(&birlesik_veri, boyut);

// Ana bilgisayar veya cihazdan erişim
// Açık transfer gerekmiyor
cekirdek<<<izgara, blok>>>(birlesik_veri);

// Birleşik belleği serbest bırak
cudaFree(birlesik_veri);
```

Akış İşleme

Eşzamanlı Yürütme

```
cudaStream_t akis1, akis2;
cudaStreamCreate(&akis1);
cudaStreamCreate(&akis2);

// Farklı akışlarda asenkron işlemler
cekirdek1<<<izgara, blok, 0, akis1>>>(veri1);
cekirdek2<<<izgara, blok, 0, akis2>>>(veri2);

cudaStreamSynchronize(akis1);
cudaStreamSynchronize(akis2);

cudaStreamDestroy(akis1);
cudaStreamDestroy(akis2);
```

Dinamik Paralellik

İç İçe Çekirdek Başlatma

```
__global__ void cocuk_cekirdek(float* veri) {
    // Çocuk çekirdek kodu
}

__global__ void ebeveyn_cekirdek(float* veri) {
    if(threadIdx.x == 0) {
        cocuk_cekirdek<<<izgara, blok>>>(veri);
        cudaDeviceSynchronize();
    }
}
```

Laboratuvar Alıştırması

Görevler

1. Paylaşımlı bellek ile matris çarpımı uygulama
2. Global bellek versiyonu ile performans karşılaştırması
3. Bellek erişim desenlerini analiz etme
4. Farklı GPU mimarileri için optimize etme

Performans Metrikleri

- Çalışma süresi
- Bellek verimi
- Doluluk oranı
- Önbellek isabet oranı

Kaynaklar

Dokümantasyon

- CUDA C++ Programlama Kılavuzu
- CUDA En İyi Uygulamalar Kılavuzu
- GPU Hesaplama Webinarları

Araçlar

- Nsight Compute
- CUDA Profilleyici
- Visual Studio GPU Hata Ayıklayıcı

Sorular ve Tartışma