CEN310 Paralel Programlama

Hafta-10 (Paralel Algoritma Tasarımı ve GPU Temelleri)

Bahar Dönemi, 2024-2025

Genel Bakış

Konular

1. Paralel Algoritma Tasarım Stratejileri
2. Ayrıştırma Teknikleri
3. GPU Mimarisi Temelleri
4. CUDA Programlamaya Giriş

Hedefler

- Paralel algoritma tasarım prensiplerini anlamak
- Veri ayrıştırma yöntemlerini öğrenmek
- GPU mimarisini keşfetmek
- CUDA programlamaya başlamak

- Görev paralelliği
- Veri paralelliği
- Boru hattı paralelliği
- Böl ve yönet

Örnek: Matris Çarpımı

```
// Sıralı versiyon
void matris_carpimi(float* A, float* B, float* C, int N) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            float toplam = 0.0f;
            for(int k = 0; k < N; k++) {
                toplam += A[i*N + k] * B[k*N + j];
            }
            C[i*N + j] = toplam;
        }
    }
}

// Paralel versiyon
#pragma omp parallel for collapse(2)
void paralel_matris_carpimi(float* A, float* B, float* C, int N) {
    for(int i = 0; i < N; i++) {
        for(int j = 0; j < N; j++) {
            float toplam = 0.0f;
            for(int k = 0; k < N; k++) {
```

Veri Ayrıştırma

- Blok ayrıştırma
- Döngüsel ayrıştırma
- Blok-döngüsel ayrıştırma

Örnek: Dizi İşleme

```
// Blok ayrıştırma
void blok_ayristirma(float* veri, int boyut, int blok_sayisi) {
    int blok_boyutu = boyut / blok_sayisi;
    #pragma omp parallel for
    for(int b = 0; b < blok_sayisi; b++) {
        int baslangic = b * blok_boyutu;
        int bitis = (b == blok_sayisi-1) ? boyut : (b+1) * blok_boyutu;
        for(int i = baslangic; i < bitis; i++) {
            // veri[i] işle
        }
    }
}
```

3. GPU Mimarisi Temelleri

Donanım Bileşenleri

- Akış Çokişlemcileri (SM'ler)
- CUDA Çekirdekleri
- Bellek Hiyerarşisi
- Warp Zamanlama

Bellek Türleri

```
CPU (Ana Bilgisayar)    GPU (Cihaz)
        ↓                    ↓
   Ana Bellek          Global Bellek
                           ↓
                    Paylaşımlı Bellek
                           ↓
                      L1 Önbellek
                           ↓
                       Yazmaçlar
```

Temel Kavramlar

- Çekirdekler
- İş Parçacıkları
- Bloklar
- Izgaralar

Merhaba Dünya Örneği

```
#include <cuda_runtime.h>
#include <stdio.h>

__global__ void merhaba_cekirdek() {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    printf("%d numaralı iş parçacığından merhaba\n", idx);
}

int main() {
    // 256 iş parçacıklı 1 blok başlat
    merhaba_cekirdek<<<1, 256>>>();
    cudaDeviceSynchronize();
```

CUDA Bellek Yönetimi

Bellek İşlemleri

```
// Cihaz belleği ayır
float *d_veri;
cudaMalloc(&d_veri, boyut * sizeof(float));

// Veriyi cihaza kopyala
cudaMemcpy(d_veri, h_veri, boyut * sizeof(float),
           cudaMemcpyHostToDevice);

// Sonuçları geri kopyala
cudaMemcpy(h_sonuc, d_sonuc, boyut * sizeof(float),
           cudaMemcpyDeviceToHost);

// Cihaz belleğini serbest bırak
cudaFree(d_veri);
```

Vektör Toplama Örneği

```
__global__ void vektor_topla(float* a, float* b, float* c, int n) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < n) {
        c[idx] = a[idx] + b[idx];
    }
}

int main() {
    int N = 1000000;
    size_t boyut = N * sizeof(float);

    // Ana bilgisayar belleği ayır
    float *h_a = (float*)malloc(boyut);
    float *h_b = (float*)malloc(boyut);
    float *h_c = (float*)malloc(boyut);

    // Dizileri başlat
    for(int i = 0; i < N; i++) {
        h_a[i] = rand() / (float)RAND_MAX;
        h_b[i] = rand() / (float)RAND_MAX;
    }

    // Cihaz belleği ayır
    float *d_a, *d_b, *d_c;
    cudaMalloc(&d_a, boyut);
    cudaMalloc(&d_b, boyut);
    cudaMalloc(&d_c, boyut);

    // Cihaza kopyala
    cudaMemcpy(d_a, h_a, boyut, cudaMemcpyHostToDevice);
    cudaMemcpy(d_b, h_b, boyut, cudaMemcpyHostToDevice);

    // Çekirdeği başlat
    int blokBasinaIs = 256;
    int izgaraBasinaBlok = (N + blokBasinaIs - 1) / blokBasinaIs;
    vektor_topla<<<izgaraBasinaBlok, blokBasinaIs>>>(d_a, d_b, d_c, N);

    // Sonucu geri kopyala
    cudaMemcpy(h_c, d_c, boyut, cudaMemcpyDeviceToHost);

    // Temizlik
    cudaFree(d_a);
    cudaFree(d_b);
    cudaFree(d_c);
    free(h_a);
    free(h_b);
    free(h_c);

    return 0;
}
```

Laboratuvar Alıştırması

Görevler

1. CUDA kullanarak matris çarpımı uygulama
2. CPU versiyonu ile performans karşılaştırması
3. Farklı blok boyutları ile denemeler
4. Bellek erişim desenlerini analiz etme

Performans Analizi

- Profil çıkarmak için nvprof kullanımı
- Çalışma süresini ölçme
- Hızlanmayı hesaplama
- Bellek transferlerini izleme

Kaynaklar

Dokümantasyon

- CUDA Programlama Kılavuzu
- CUDA En İyi Uygulamalar Kılavuzu
- NVIDIA Geliştirici Blogu

Araçlar

- NVIDIA NSight
- CUDA Araç Seti
- Görsel Profilleyici

Sorular ve Tartışma