Praktikum 03

IF3230 - Sistem Paralel dan Terdistribusi

CUDA - Bitonic Sort

Dipersiapkan oleh:
Asisten Laboratorium Sistem Terdistribusi

Didukung oleh:

Start Date: 23 Maret 2018

End Date: 29 Maret 2018

A. Persiapan Praktikum

Pada praktikum kali ini anda ditugaskan untuk menggunakan CUDA, perkakas untuk melakukan pemrograman secara paralel yang berbasiskan GPGPU (General-Purpose GPU). Server yang dapat dipakai untuk menggunakan CUDA adalah **167.205.32.236**.
Anda dapat mengakses server menggunakan username **NIM** dan password **'guest'**.

B. Pengenalan CUDA

CUDA (**Compute Unified Device Architecture**) adalah standar yang dibuat oleh NVidia untuk melakukan pemrograman GPU pada Graphic Card NVidia. Penjelasan CUDA yang lebih lengkap dapat dilihat di slide yang diunggah di gitlab. Sebelum membahas lebih lanjut mengenai pemrograman CUDA, berikut ini adalah istilah-istilah dalam pemrograman CUDA:

1. Host: CPU.
2. Device: GPU.
3. Kernel: Kode yang berjalan di atas GPU. Satu GPU hanya dapat menjalankan satu kernel pada satu saat.
4. Thread: Satuan eksekusi pada GPU. Terdapat banyak thread yang menjalankan kernel secara bersamaan.
5. Block: Kumpulan Thread. Block merupakan satuan sinkronisasi eksekusi. Satu block tidak dapat berkoordinasi dengan block lainnya.
6. Grid: Kumpulan block.

Eksekusi Kernel

Sebuah kernel akan dijalankan oleh sejumlah *thread*. Setiap *thread* akan menjalankan kernel yang sama. Setiap *thread* akan mendapatkan ID unik yang dapat dipakai untuk menentukan alur eksekusi *thread*. *Thread* terkelompok menjadi satuan yang disebut *block*. *Thread* yang berada pada satu *block* yang sama dapat melakukan koordinasi yang lebih lanjut seperti melakukan *sharing memory* dan sinkronisasi.

Arsitektur Memory dan Hardware

Setiap *thread* memiliki akses ke ***register***, ***shared memory***, dan ***device memory***. Setiap *thread processor* memiliki *register*, *register* tersebut hanya dapat diakses oleh thread processor tersebut. Setiap *block* memiliki *shared memory* yang dapat diakses oleh setiap *thread* pada *block* tersebut. *Device memory* dapat diakses oleh semua *thread* pada *block* manapun. Secara hardware, *thread* berjalan di *thread processor*. *Block* berjalan di atas ***streaming multiprocessor* (SM)**. Satu SM dapat menjalankan banyak *block* sekaligus. Hal ini dapat mempengaruhi ukuran *shared memory* yang dapat digunakan oleh setiap *block*.

Pemrograman CUDA

CUDA menambahkan beberapa ***sintaks***, ***built-in variables***, dan ***runtime functions***:

1. *Function type qualifier*: membedakan fungsi yang dieksekusi di host dan di device.

    a. **__device__**: dieksekusi di device, dapat dipanggil melalui device. (fungsi internal yang digunakan oleh kernel.)
    b. **__global__**: dieksekusi di device, hanya dapat dipanggil oleh host. (berfungsi sebagai titik awal eksekusi kernel di device).
    c. **__host__**: dieksekusi di host, hanya dapat dipanggil oleh host.
2. *Variable type qualifier*: dipakai untuk mendeklarasikan sifat variabel sehingga diletakkan di lokasi memori yang sesuai.
3. *Built-in variables*: variable yang terdefinisi secara otomatis di saat runtime.
    a. **gridDim**: variabel yang berisi dimensi dari grid.
    b. **blockIdx**: variabel yang berisi index block di mana thread ini berada.
    c. **blockDim**: variabel yang berisi dimensi dari block.
    d. **threadIdx**: variabel yang berisi index thread di dalam block. (untuk membedakan thread yang berada di block yang berbeda, gunakan blockIdx).
4. Parameter eksekusi kernel: kernel diseksekusi dengan memanggil fungsi **__global__** dengan memberikan nilai `<<<grid, block>>>` tepat di belakang nama fungsi yang ingin dieksekusi (sebelum '('). `grid` dan `block` inilah yang akan menjadi **gridDim** dan **blockDim** di saat berjalannya kernel.
5. *Runtime functions*: fungsi yang mengatur alur eksekusi fungsi. (contoh: **__syncthreads()**).

Eksekusi dan Contoh Program

Kompilasi program CUDA dapat dilakukan dengan menggunakan perintah ini:

```
nvcc <file-name>.cu -o <executable-name>
```

Jalankan perintah di bawah ini untuk menjalankan program hasil kompilasi:

```
./<executable-name>
```

Pada *repository project* terdapat beberapa contoh program yang dapat dijalankan, silakan gunakan program tersebut sebagai acuan.

C. Spesifikasi Tugas

Pada tugas ini, anda diminta untuk mengimplementasikan *bitonic sort* secara paralel pada CUDA. Bitonic Sort adalah algoritma *sorting* berbasis perbandingan yang dapat dilakukan secara paralel. Algoritma ini mengubah sekuens angka menjadi *bitonic sequence*, yaitu sekuens yang bertambah dan sekuens yang berkurang.

Sebagai contoh, berikut merupakan source code bitonic sort dalam bahasa C yang dijalankan secara sekuensial.

Untuk mengukur apakah paralelisasi bitonic sort berhasil dilakukan, anda diminta untuk melakukan uji kinerja (*performance*) pada program yang anda buat. Pengujian kinerja dilakukan dengan metode sebagai berikut:

1. Inisialisasi array of integer berisi **N** elemen. Setiap elemen berisi nilai random yang di-generate dari rand(). Gunakan seed yang sama untuk setiap pembangkitan array. Fungsi pembangkitan array akan disediakan oleh asisten.

   Gunakan fungsi di bawah untuk membangkitkan array.

```
void rng(int* arr, int n) {
    int seed = 13515000; // Ganti dengan NIM anda sebagai seed.
    srand(seed);
    for(long i = 0; i < n; i++) {
        arr[i] = (int)rand();
    }
}
```

2. Terapkan bitonic sort pada array sehingga array terurut dari kecil ke besar.
3. Pengukuran waktu dilakukan pada saat bitonic sort dimulai hingga selesai. Jangan masukkan pembangkitan array pada pengukuran waktu.
4. Gunakan *microsecond* sebagai satuan dasar pada perhitungan waktu yang anda gunakan.

Pada pengujian ini, **N** yang digunakan adalah **512**, **1.024**, **4.096**, **65.536**, **262.144**, **1.048.576**, dan **8.388.608**. Agar pengujian lebih akurat, jalankan setiap kasus uji setidaknya tiga kali. Tuliskan hasil pengujian anda pada laporan pengerjaan yang berisi:

- Deskripsi solusi paralel. Berikan ilustrasi jika perlu.
- Analisis solusi yang anda berikan. Apakah mungkin terdapat solusi yang memberikan kinerja lebih baik?
- Jumlah thread yang digunakan. Kenapa anda memilih angka tersebut?
- Pengukuran kinerja untuk tiap kasus uji (jumlah N pada array) dibandingkan dengan bitonic sort serial.
- Analisis perbandingan kinerja serial dan paralel.

D. Pengumpulan dan Deliverables

Tugas dikerjakan dalam kelompok sebanyak maksimal 2 orang (anggota kelompok tidak boleh dari kelas yang berbeda). Fork spesifikasi tugas ini serta contoh source code CUDA dari *repository* http://gitlab.informatika.org/IF3230-2018/CUDA. Kerjakan persoalan yang diberikan pada deskripsi di atas maksimal **29 Maret 2018** pukul **23:59 WIB**.

Lakukan merge request pada repository awal paling lambat pada waktu dan tanggal yang sama. Merge request dilakukan dengan judul **Praktikum3_K0[1|2|3]_<NIM1>_<NIM2>.** Perhatikan bahwa **keterlambatan pengumpulan dapat mengakibatkan nilai 0 (nol)**.

Beberapa file yang harus ada dalam *repository* tersebut diantaranya:

- Direktori **src** yang berisi *source code* yang anda buat.

- File **output** yang berisi hasil uji bitonic sort pada data uji. Contoh output serta data uji akan diberikan pada repository gitlab.
- **Makefile**. Buatlah sehingga kompilasi program dapat dilakukan hanya dengan pemanggilan command `'make'` saja.
- File **README.md** yang berisi:
    - Petunjuk penggunaan program.
    - Pembagian tugas. Sampaikan dalam list pengerjaan untuk setiap mahasiswa. Sebagai contoh: XXXX mengerjakan fungsi YYYY, ZZZZ, dan YYZZ.
    - Laporan pengerjaan, dengan struktur laporan sesuai dengan deskripsi pada bagian sebelumnya.

Segala bentuk kecurangan yang terjadi akan ditindaklanjuti oleh asisten dan dikenakan konsekuensi.