SKRIPSI

PENYELESAIAN MASALAH KNAPSACK DENGAN MENGGUNAKAN MULTI-PROCESSOR PADA GPU

Samuel Lusandi

NPM: 2014730001

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2018

UNDERGRADUATE THESIS

SOLVING KNAPSACK PROBLEM USING THE MULTIPROCESSORS OF A GPU

Samuel Lusandi

NPM: 2014730001

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2018

LEMBAR PENGESAHAN

PENYELESAIAN MASALAH KNAPSACK DENGAN MENGGUNAKAN MULTI-PROCESSOR PADA GPU

Samuel Lusandi

NPM: 2014730001

Bandung, 22 Mei 2018

Menyetujui,

Pembimbing

Joanna Helga, M.Sc.

Ketua Tim Penguji

Dott. Thomas Anung Basuki

Anggota Tim Penguji

Elisati Hulu, M.T.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENYELESAIAN MASALAH KNAPSACK DENGAN MENGGUNAKAN MULTI-PROCESSOR PADA GPU

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhkan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 22 Mei 2018

Samuel Lusandi
NPM: 2014730001

ABSTRAK

Dewasa ini, perkembangan *Central Processing Unit* (CPU) mengarah pada banyaknya inti pemroses dan komputasi paralel. Meskipun begitu, jumlah inti pada CPU dengan harga yang cukup terjangkau hanya mencapai 4 hingga 8 saja. Semenjak tahun 2006, NVIDIA telah membuat perangkat *Graphical Processing Unit* (GPU) yang mereka rakit mampu membantu perhitungan CPU dengan teknologi bernama CUDA. Bila dibandingkan dengan CPU, GPU memiliki inti perhitungan yang banyak namun kecepatan perhitungannya rendah dan hanya mendukung operasi aritmatika sederhana. Sebaliknya, CPU yang memiliki inti sedikit memiliki kecepatan yang tinggi dan dapat melakukan perhitungan dan instruksi kompleks.

Dalam penelitian ini, akan dibahas sebuah permasalahan klasik dalam dunia pemrograman, yakni masalah *knapsack*. Diberikan sejumlah barang yang masing-masing memiliki berat dan nilai jual serta sebuah tas dengan kapasitas berat tertentu. Tujuan dari masalah ini adalah mencari sejumlah barang dengan total berat tidak melebihi kapasitas tas tersebut, namun menghasilkan nilai jual yang terbesar. Permasalahan ini dapat diselesaikan dengan algoritma *Dynamic Programming* dan akan diimplementasikan secara paralel dengan bantuan GPU.

Implementasi dari algoritma ini telah dibangun dengan bantuan GPU untuk mempercepat perhitungan. Meskipun begitu, terdapat bagian-bagian yang masih menggunakan CPU dalam prosesnya. Pengujian perangkat dilakukan dengan kasus yang telah dihitung dan kasus acak. Pengujian percepatan yang diperoleh juga diuji untuk beberapa kondisi tertentu, seperti jumlah barang dan besarnya kapasitas tas yang harus diperhitungkan. Untuk komputasi yang sedikit, algoritma ini mengalami degradasi perhitungan yang sangat besar karena *cost* interaksi CPU dengan GPU yang besar. Sebaliknya, untuk perhitungan yang besar, perhitungan yang dilakukan di GPU mengalami percepatan yang sangat tinggi, mencapai hingga sepuluh kali lipat.

Kata-kata kunci: GPU, komputasi paralel, permasalahan *knapsack*, *Dynamic Programming*

ABSTRACT

In recent years, Central Processing Unit (CPU) development has moved to having multiple cores and in turns, parallel computing. However, consumer CPUs with acceptable prices only reach around 4 to 8 cores. At the same time, ever since NVIDIA introduced CUDA for Graphical Processing Units (GPUs) in 2006, GPUs have been a choice to accelerate calculations that were too heavy for CPUs. Compared to CPUs, GPUs have a lot more cores but have slow core speed and are only able to do simple arithmetic calculations. CPUs, on the other hand, have faster core speed and are capable of complex instructions despite having less compute cores.

This research uses one of the classical programming problems known as the Knapsack problem. The Knapsack problem aims to find a subset of items with certain values and weights from a set that can fit in a knapsack of a certain capacity while getting the maximum value out of it. This problem is solved using Dynamic Programming and will be implemented with the help of GPU in parallel.

The implementation of this algorithm has been built with the help of GPU to accelerate the calculations. Despite this, most of the sections of the calculations are done in the CPU. The software has been tested with calculated and random cases. The speedup results have been tested with varying number of items and the capacity of the knapsack that must be calculated. For smaller computation (i.e. when the capacity is small), the GPU and CPU interaction cost outweighs the benefits of the speedup, making the performance degrade significantly. Conversely, large computational space shows that the GPU accelerated calculation results in an enormous increase in performance that reaches more than ten times in speedup.

Keywords: GPU, parallel computing, Knapsack problem, Dynamic Programming

Dipersembahkan untuk ibu saya dan almarhum ayah saya

KATA PENGANTAR

Puji dan syukur kepada Tuhan Yesus Kristus atas kasih karunia, anugerah, dan pimpinannya sehingga penulis dapat menyelesaikan skripsi berjudul "Penyelesaian Masalah Knapsack dengan Menggunakan Multi-Processor pada GPU". Hambatan dan kendala dihadapi penulis, namun penyertaan-Nya memampukan penulis untuk menyelesaikan skripsi ini tepat pada waktunya. Penyelesaian skripsi ini juga terjadi dengan bantuan dari berbagai pihak dalam bentuk yang berbeda-beda. Oleh sebab itu, penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Tuhan Yesus Kristus yang memimpin pengerjaan skripsi ini dan selama perkuliahan di Universitas Katolik Parahyangan.
2. Ibu penulis yang berjerih payah melakukan pekerjaan untuk membiayai perkuliahan penulis dan memberi dukungan penuh selama perkuliahan dalam berbagai bentuk.
3. Ibu Joanna Helga selaku pengajar, pelatih, dan pembimbing skripsi penulis selama perkuliahan yang telah banyak mengorbankan waktu untuk mendidik dan mengajarkan banyak hal yang tidak diajarkan di perkuliahan.
4. Keenan dan Albert selaku teman satu tim kompetisi ACM dan menjadi teman baik selama perkuliahan di UNPAR yang mendukung penulis dalam banyak hal.
5. Bapak Chandra Wijaya selaku ketua Laboratorium FTIS selama dua semester penulis menjadi Administrator dan telah mengajarkan banyak hal tentang jaringan.
6. Ibu Luciana Abednego selaku ketua Laboratorium FTIS selama dua semester terakhir penulis menjadi Administrator FTIS.
7. Chris selaku teman bertukar pikiran dalam banyak hal selama perkuliahan penulis.
8. Seluruh dosen Teknik Informatika UNPAR yang telah mengajar dan memberikan ilmunya kepada penulis.
9. Rekan-rekan Administrator Laboratorium FTIS yang telah menjadi kolega dan teman seperjuangan selama penulis menimba ilmu di UNPAR.
10. Teman-teman dari Nilem Studio yang telah membantu secara moril dan memotivasi penyelesaian skripsi penulis.
11. Aku Nono yang setiap hari untuk tujuh semester mengantarkan penulis dari rumah ke kampus.
12. Marcell, Kalas, Hereza, Vincent, Topher, dan masih banyak teman-teman dan pihak-pihak lain yang telah membantu penulis selama perkuliahan.

Semoga skripsi penulis dapat menjadi manfaat untuk penelitian-penelitian berikutnya. Bila terdapat kesalahan dalam penulisan skripsi ini, penulis mohon maaf sebesar-besarnya.

Bandung, Mei 2018

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Kartu grafik akhir-akhir ini sering digunakan untuk melakukan perhitungan-perhitungan intensif dalam suatu program atau fungsi. Hal ini dikarenakan banyaknya inti perhitungan yang dimiliki oleh setiap kartu grafik jauh melampaui jumlah inti yang dimiliki *processor* pada umumnya. *Processor* atau CPU (*Central Processing Unit*) pada umumnya dengan harga yang terjangkau hingga saat ini memiliki antara 1 hingga 8 inti dengan teknologi yang membuat mereka memiliki dua kali jumlah inti asli mereka (menjadikan mereka memiliki 2 hingga 16 *inti virtual*) yang dikenal sebagai teknologi *hyperthreading*. Kartu grafik atau umumnya dikenal sebagai GPU (*Graphical Processing Unit*) dapat memiliki puluhan hingga ratusan kali lebih banyak unit perhitungan dibandingkan *CPU* dengan harga yang sama. CPU pada saat dokumen ini ditulis dengan jumlah inti terbanyak hanya memiliki 28 inti dan 56 dengan *hyperthreading*, memiliki harga belasan kali lebih mahal dari GPU dengan jumlah inti paling banyak saat ini (yakni 3840 inti) sehingga bukan menjadi pilihan untuk digunakan pengguna secara umum.

Untuk perhitungan-perhitungan berskala besar tidak memiliki ketergantungan data, NVIDIA, sebuah perusahaan GPU, memperkenalkan CUDA (*Compute Unified Device Architecture*) pada tahun 2006. CUDA adalah nama yang diberikan oleh NVIDIA kepada unit pemroses dalam GPU rakitannya yang dapat digunakan untuk melakukan perhitungan yang bersifat umum (*general-purpose computation*). Dengan begitu banyak inti dalam satu buah GPU, perbedaan performansi dari CPU dan GPU cukup besar bila perhitungan tersebut dilakukan secara paralel.

Setiap inti GPU memiliki kecepatan yang cukup kecil dibandingkan dengan CPU pada umumnya. Secara umum, GPU memiliki inti dengan kecepatan 1 hingga 2 GHz, sementara CPU pada umumnya memiliki kecepatan 2 hingga 4 GHz. Hal ini membuat perhitungan-perhitungan yang dilakukan secara sekuensial (hanya menggunakan satu inti atau unit pemroses) pada GPU berjalan lebih lambat.

Permasalahan *knapsack* adalah permasalahan klasik di dunia ilmu komputer. Diberikan sebuah kantung yang dapat menampung sejumlah barang dengan beban tertentu. Diketahui juga bahwa terdapat sejumlah barang dengan berat dan harga jualnya masing-masing. Tujuan dari permasalahan *knapsack* ini adalah untuk memaksimalkan harga jual dari barang-barang yang dimasukkan ke dalam kantung tersebut, tanpa melampaui batas berat yang dapat ditampung oleh kantung tersebut. Sebagai catatan, barang yang dipilih harus dimasukkan secara menyeluruh, atau tidak sama sekali (suatu barang tidak dapat dimasukkan sebagian saja).

Permasalahan *knapsack* ini dapat diselesaikan dengan beberapa cara mendasar, yakni dengan mencoba semua kemungkinan (dikenal dengan *brute force*) dan menggunakan teknik rasio berat terhadap nilai barang yang dikenal sebagai teknik *greedy*. Untuk permasalahan *knapsack* di mana barang hanya dapat diambil atau tidak (tidak dapat diambil sebagian saja), teknik *greedy* belum tentu membuahkan hasil yang terbaik. Terdapat beberapa kasus di mana teknik ini mengambil barang-barang yang justru menutup kemungkinan pengambilan barang-barang lain yang dapat menghasilkan nilai yang terbaik.

Teknik *brute force* sendiri merupakan teknik yang sudah pasti menghasilkan nilai yang paling

optimal karena sifatnya yang mencoba semua kemungkinan pengambilan barang. Namun, kompleksitas perhitungannya sangat tinggi sehingga akan memakan waktu yang sangat lama ketika skala permasalahan (jumlah barang yang dapat diambil) bertambah banyak. Analisis menunjukkan bahwa dalam teknik ini, terdapat banyak kondisi-kondisi yang sudah pernah dihitung sebelumnya dihitung kembali. Masalah ini dikenal sebagai *overlapping subproblems*.

Masalah dari teknik *brute force* ini dapat dioptimasi dengan menggunakan teknik *Dynamic Programming*. Teknik *Dynamic Programming* adalah teknik yang bekerja dengan mencatat informasi perhitungan yang sudah pernah dilakukan, sehingga informasi tersebut dapat digunakan lagi bilamana terjadi perhitungan serupa. Dengan begitu, kompleksitas waktu perhitungan teknik *brute force* dapat diperkecil. Meksipun begitu, teknik ini memiliki kompleksitas waktu maupun tempat yang besar, yakni $O(NW)$ di mana N menyatakan jumlah barang yang tersedia untuk dipilih dan W menyatakan kapasitas kantung *knapsack* yang dimiliki. Karena kompleksitas yang cukup besar, perhitungan yang dilakukan secara sekuensial masih memakan waktu yang cukup lama ketika data yang diberikan besar.

Dengan perkembangan komputer yang semakin maju dalam bidang komputasi paralel, algoritma *Dynamic Programming* untuk menyelesaikan masalah *knapsack* ini kemudian dikembangkan untuk menggunakan inti-inti yang tersedia pada *CPU* secara paralel. Dengan jumlah inti yang jauh lebih banyak, diharapkan kinerja dari algoritma ini dapat ditingkatkan lebih jauh lagi dengan menggunakan inti yang tersedia pada GPU.

Dalam melakukan perhitungan dengan bantuan GPU, CPU membutuhkan waktu interaksi yang sangat tinggi dengan GPU. Waktu interaksi ini dapat menyebabkan waktu perhitungan yang dilakukan dengan bantuan GPU melampaui waktu perhitungan yang dilakukan pada CPU konvensional. Karena itu, dalam penelitian ini, akan diterapkan algoritma-algoritma kompresi yang dapat membantu mengurangi interaksi yang dilakukan antara CPU dengan GPU. Ini bertujuan untuk memastikan kinerja dari perangkat yang dibangun maksimal.

1.2 Rumusan Masalah

Berikut adalah rumusan masalah yang akan dibahas dalam skripsi ini:

1. Bagaimana cara kerja GPU dibandingkan CPU pada umumnya?
2. Bagaimana cara CPU berkomunikasi dengan GPU dan seberapa besar *cost* interaksi tersebut?
3. Bagaimana mengimplementasikan algoritma *Dynamic Programming* untuk *knapsack* secara paralel?
4. Bagaimana mengimplementasikan kode yang dapat menggunakan bantuan GPU dan bukan hanya inti pada CPU?
5. Bagaimana cara untuk melakukan kompresi data agar interaksi antara CPU dengan CUDA optimal?
6. Seberapa besar perbedaan kinerja antara GPU berbasis CUDA dengan CPU pada umumnya dalam menyelesaikan permasalahan *knapsack*?

1.3 Tujuan

Tujuan yang hendak dicapai dalam pembuatan skripsi ini adalah:

1. Mempelajari cara kerja GPU dalam melakukan perhitungan bila dibandingkan dengan cara kerja CPU pada umumnya.

2. Mempelajari bagaimana CPU dan GPU berinteraksi dan seberapa besar *cost* interaksi tersebut bila dibandingkan dengan keuntungan performansi yang diperoleh.
3. Mengimplementasikan algoritma *Dynamic Programming* untuk masalah *knapsack* secara paralel di GPU.
4. Membuat perangkat lunak untuk menyelesaikan masalah *knapsack* dengan menggunakan algoritma *Dynamic Programming* secara paralel menggunakan GPU berbasis CUDA.
5. Melakukan kompresi dan optimasi terhadap komunikasi antara CPU dengan GPU sehingga waktu untuk melakukan interaksi antar-perangkat tidak menghambat waktu perhitungan yang sesungguhnya.
6. Melakukan perbandingan antara kecepatan perhitungan yang dilakukan pada CPU dan pada GPU dalam menyelesaikan permasalahan Knapsack.

1.4 Batasan Masalah

Dalam pengerjaannya, skripsi ini hanya akan membahas tentang hal-hal berikut ini:

1. Knapsack 0/1 sesuai yang telah dijelaskan pada Bagian 1.1,
2. Penggunaan GPU berbasis CUDA untuk membantu perhitungan perhitungan Knapsack.

Variabel-variabel yang akan diatur dalam melakukan eksperimen dalam skripsi ini adalah:

1. Variasi besarnya jumlah barang dan kapasitas *knapsack*,
2. Kecepatan inti dan memori GPU untuk menguji seberapa besar dampaknya terhadap waktu perhitungan.

1.5 Metodologi

Metodologi penelitian yang akan digunakan antara lain:

1. Melakukan studi pustaka mengenai penyelesaian masalah Knapsack dengan algoritma Dynamic Programming.
2. Melakukan studi pustaka mengenai implementasi algoritma Dynamic Programming untuk masalah Knapsack dengan menggunakan GPU berbasis CUDA.
3. Melakukan studi pustaka mengenai arsitektur dan model memori GPU.
4. Melakukan pembelajaran untuk melakukan kompresi data yang akan ditransfer dari CPU ke GPU untuk mengoptimasi kinerja perhitungan algoritma ini.

1.6 Sistematika Penulisan

Tulisan ini disusun dalam enam bab yang akan berisi:

1. Bab 1 Pendahuluan
   Bab 1 berisi membahas tentang latar belakang mengenai CPU, GPU, dan bagaimana GPU dapat membantu perhitungan *knapsack* dengan algoritma Dynamic Programming. Pada bab ini juga dibahas tentang rumusan masalah, tujuan yang ingin dicapai, batasan masalah, metodologi penelitian, dan sistematika penulisan penelitian ini.

2. Bab 2 Landasan Teori
   Bab 2 berisi dasar teori yang berhubungan dengan algoritma *Dynamic Programming* untuk penyelesaian masalah Knapsack, arsitektur GPU berbasis CUDA secara umum, dan bagaimana menggunakan bantuan GPU untuk menyelesaikan masalah *knapsack* secara paralel. Dibahas pula bagaimana untuk mengoptimasi interaksi dan transaksi antara CPU dengan GPU agar keuntungan performansi yang diperoleh maksimal.

3. Bab 3 Analisis
   Bab 3 berisi analisis yang berhubungan dengan implementasi algoritma *knapsack* dengan bantuan GPU. Pembahasan pada bab ini mencakup tipe memori yang akan digunakan, jumlah *thread*, dan tipe-tipe data yang akan digunakan untuk memastikan performansi yang optimal.

4. Bab 4 Perancangan
   Bab 4 berisi perancangan perangkat lunak dan pengujian yang akan dilakukan untuk memastikan kebenaran program dan menunjukkan keuntungan performansi perhitungan yang dilakukan dengan bantuan GPU.

5. Bab 5 Implementasi dan Pengujian
   Bab 5 berisi implementasi perangkat lunak dan pengujian perangkat lunak. Disertakan juga kesimpulan hasil pengujian yang telah dilakukan dalam implementasi perangkat lunak.

6. Bab 6 Kesimpulan dan Saran
   Bab 6 berisi kesimpulan penelitian dan saran untuk lebih lanjut mengembangkan perangkat lunak berdasarkan penelitian ini.