Pengaruh Paralelisme Terhadap Mesin Turing Sebagai Konsep Komputasi

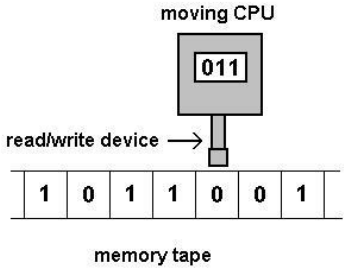
Fitrandi Ramadhan and 23515050
Program MagisterInformatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
fitrandiramadhan@gmail.com

***Abstract*—Mesin turing non-deterministik adalah mesin turing yang dapat memberikan aksi yang tidak unik terhadap sebuah state yang diterima. Hal ini dapat searah dengan perkembangan teknologi komputer saat ini yaitu multiprocessing atau parallelism. Pertanyaan muncul apakah hal ini perkembangan paralelisme memang berpengaruh terhadap non-deterministik process dari non-deterministic algorithm. Dengan perubahan ini diharapkan dapat merubah pemahaman computer scientist terhadap analisis sebuah algoritma dimana paralelisme menjadi sebuah paradigma yang berdiri sendiri dikarenakan paralelisme sangat berbeda dengan singular programming.**

***Index Terms*—non-deterministic, turing machine, parallelism, computing.**

I. PENDAHULUAN

Dalam perkembangan komputer kita tidak pernah terlepas dari mesin turing sebagai sebuah konsep bahwa sebuah mesin tanpa kecerdasan natural dapat menyelesaikan sebuah persoalan jika diberikan aturan-aturan tertentu. Sebuah mesin dapat menyelesaikan persoalan sesuai dengan aturan yang telah diberikan, tentu kita berfikir hal ini adalah hal yang mudah dengan mendefinisikan aturan paling sederhana yaitu aturan if-then. Mesin turing dapat menyelesaikan masalah yang sudah didefinisikan dan tidak dapat menyelesaikan masalah yang tidak didefinisikan pada mesin turing tersebut



Gambar 1.1 Single Tape Turing Machine.

Secara natural pula apabila berada pada suatu situasi mesin akan melakukan satu dan hanya satu aksi. Hal ini berbeda dengan proses berfikir manusia yang dapat melakukan lebih dari satu aksi terhadap situasi yang diberikan. Fenomena ini disebut proses non-deterministic. Pertanyaan yang muncul dari fenomena ini adalah apa yang terjadi apabila sebuah mesin dapat melakukan proses berfikir ini. Hipotesis yang diajukan adalah dengan proses berfikir mesin dapat menyelesaikan lebih banyak permasalahan atau permasalahan yang lebih kompleks daripada deterministic turing machine. Waktu yang diperlukan untuk menyelesaikan permasalahan yang sama yang diberikan pada deterministic turing machine pun dapat diselesaikan dengan lebih cepat.

II. LANDASAN TEORI

Secara definisi non deterministic turing machine adalah mesin turing yang dapat memiliki lebih dari satu aksi terhadap situasi yang diberikan. Berikut elemen dari fungsi transisi yang ada pada deterministic turing machine.

1. State yang dibaca pada pita.
2. Simbol yang akan ditulis pada pita
3. Arah dari head setelah melakukan aksi

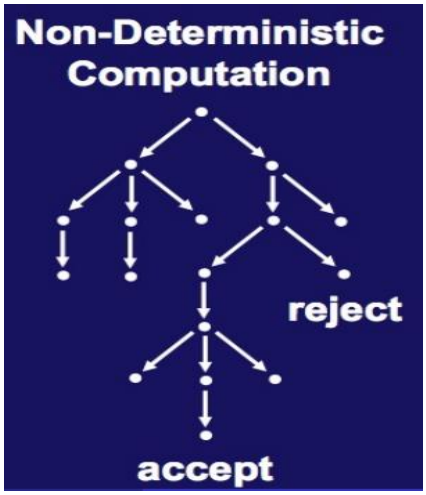
Hal yang sama juga dimiliki oleh non-deterministic turing machine, akan tetapi mesin dapat memiliki lebih dari satu poin kedua dan ketiga. Berikut definisi formal sebuah non-deterministic turing machine.

Sebuah mesin turing non-deterministik dapat didefinisikan secara formal sebagai 6 tuple.

$$M = (Q, \Sigma, \iota, \sqcup, A, \delta)$$

1. Q adalah state yang diberikan
2. Σ adalah alphabet
3. $\iota \in Q$ adalah initial state
4. $\sqcup \in \Sigma$ adalah symbol blank.
5. $A \subseteq Q$ adalah final state.

Ilustrasi dari non deterministic turing machine adalah sebagai berikut.



Gambar 2.1 Ilustrasi algoritma non deterministik.

Setiap varian dari mesin turing memiliki representasinya pada standard deterministic turing machine. Pembuktian dari pernyataan ini adalah sebagai berikut.

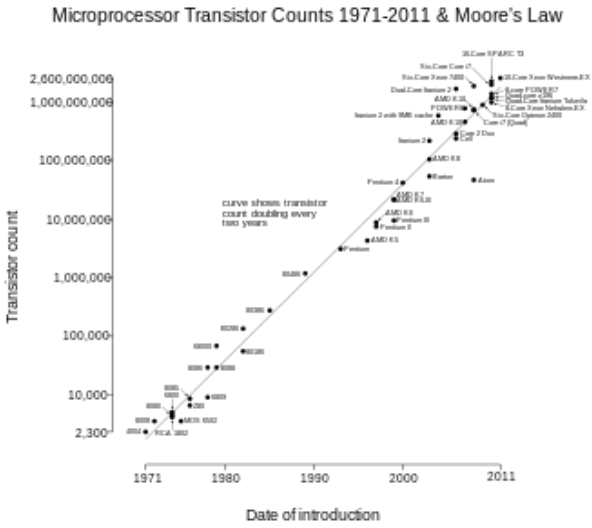
Diberikan N = (Q, Σ, Γ, Δ, q0, qa, qr), dan d bilangan bulat terkecil untuk setiap (q, σ).

1. Sebuah mesin turing deterministic menelusuri setiap cabang dari komputasi pohon yang dibangkitkan oleh N.
2. Walaupun waktu penelusuran eksponensial total waktu yang dibutuhkan terbatas (finite).
3. Pada akhirnya proses ditentukan oleh kardinalitas dari $|\delta(Q \times \Gamma)|$, yang adalah finite, karena $\delta(Q \times \Gamma) \subseteq Q \times \Gamma \times \{L, R\}$).

Problem yang diselesaikan dengan non-deterministic turing machine tentunya adalah problem-problem NP. Setiap branching pada NP diselesaikan secara non-deterministik oleh mesin dan apabila salah satu branch menerima L sebagai bahasa yang diterima maka akan dikembalikan "accept". salah satunya adalah hamiltonian circuit.

III. ANALISIS

Perkembangan perangkat keras komputer saat ini bergerak menuju penambahan jumlah core proccessor. Moore's law dan perkembanganya membuktikan bahwa sangat sulit saat ini untuk mengembangkan CPU dengan kecepatan yang lebih besar, akan tetapi perkembangan terus dilakukan untuk meningkatkan efisiensi spasial transistor pada CPU untuk menambah jumlah physical core pada CPU. Pada sisi pengembang perangkat lunak hal ini memotivasi pengembang untuk memanfaatkan jumlah core yang banyak untuk melakukan paralelisasi pada program yang dibuat. Sebuah program yang dibuat dengan paralelism akan jauh lebih cepat daripada program yang sama dengan single proccess paradigm.



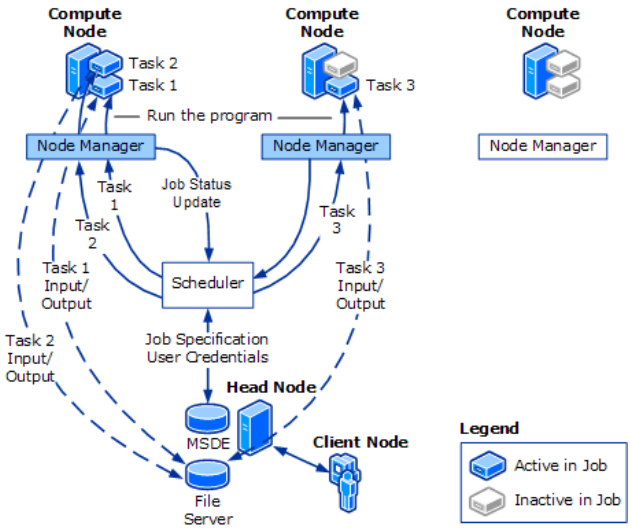
Gambar 3.1 Moore's Law dan perkembanganya.

Secara definisi sebuah algritma non-deterministik melakukan operasi percabangan proses. Pada single proccess programming setiap daun dari proses akan dikunjungi baik dengan DFS (Depth First Search) ataupun BFS (Breadth First Search). Dengan kemampuan komputer saat ini setiap percabangan ini dapat dikunjungi (diproses) secara paralel, meningkatkan efisiensi dan memanfaatkan kemampuan komputer semaksimal mungkin. Berikut kode dari sebuah program paralel.

```
//file            :     Prime.cpp
//author    :     steaKK

#include "Prime.hpp"

Prime::Prime() {
      size = DEFAULT_SIZE;
      data = new bool[size];
      for(long         i=0;i<=size;i++)
data[i] = true;
}

Prime::Prime(long _size) {
      size = _size;
      data = new bool[size];
      for(long         i=0;i<=size;i++)
data[i] = true;
}

Prime::Prime(const Prime& _Prime) {
      size = _Prime.size;
      data = new bool[size];
      for(long  i=0;i<size;i++)  data[i]
= _Prime.data[i];
}
```

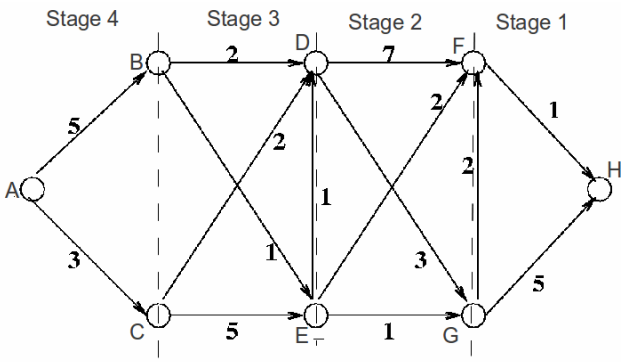
```
Prime&  Prime::operator=(const  Prime&
_Prime) {
      size = _Prime.size;
      for(long  i=0;i<size;i++)  data[i]
= _Prime.data[i];
      return *this;
}

Prime::~Prime() {

}

bool& Prime::get_data() {
      return data;
}

void    Prime::set_data(const    bool&
_data) {
      data = _data;
}

void Prime::sieve_of_erastothenes() {
      for(long   i=2;i<=(size-1)/2;i++)
{
            flip_data(i);
      }
}

void Prime::flip_data(long x) {
      #pragma omp parallel for
      for(long i=x+x;i<size;i+=x) {
            data[i] = false;
      }
}

void Prime::print() {
      for(int i=0;i<size;i++) {
            if(data[i]==true)  cout  <<
i << " is prime" << endl;
            else  cout  <<  i  <<  "  is
composite" << endl;
      }
}

void Prime::print2() {
      for(int i=0;i<size;i++) {
            if(data[i]==true)  cout  <<
"[" << i << "]";
      }
      cout << endl;
}

void Prime::print3() {
      long count = 0;
      for(int i=0;i<size;i++) {
            if(data[i]==true) count++;
      }
      cout << "n primes to " << size-1
<< " = " << count-2 << endl;
}
```

Dari kode tersebut dapat dilihat bahwa pada pengunjungan setiap cabang pohon dapat dibuat sebuah thread baru yang akan diproses oleh processor lain. Dan apabila sebuah proses selesai dan menemukan jawaban atas persoalan (accept/reject), maka thread tersebut dapat menginformasikan thread lainya untuk berhenti melakukan komputasi dan mengembalikan jawaban. Tidak hanya dilakukan pada satu buat komputer dengan banyak core proses, paralelisasi juga dapat dilakukan antar komputer dengan library seperti MPI (Message Passing Interface) setiap komputer dapat berkomunikasi satu sama lain dan memberi perintah kepada program lain untuk berhenti melakukan proses apabila sebuah jawaban telah ditemukan pada proses lainya. Lain dari perkembangan proccessor yang berkembang sangat pesat perkembangan memory berjalan sangat lambat.



Gambar 3.2 Message Passing Interface.

Sebuah metode penyelesaian masalah yaitu dynamic programming adalah sebuah metode pemanfaatan ruang untuk melakukan pencarian dari percabangan masalah. Dynamic programming menyimpan optimum solution untuk setiap state yang dilalui dan dengan paralelisasi penggunaan lebih dari satu komputer dan memanfaatkan spasial memory dari setiap komputer akan mengoptimasi spasial yang dibutuhkan untuk melakukan proses.



Gambar 3.3 Dynamic Programming

IV. PENGUJIAN

Untuk eksperimen akan diimplementasikan kode untuk eksperimen penggunaan algoritma non-deterministik dengan singular dan paralel. Program yang dibuat adalah program untuk melakukan sorting pada sebuah larik yang menimpan bilangan bulat acak. Psudocode dari program ini sendiri adalah algoritma singular akan tetapi untuk optimasi dapat dilakukan paralelisasi demi meningkatkan kecepatan program. Berikut adalah kode yang digunakan.

```
//file            :     CountSort.h
//author    :     steaKK

#ifndef CountSort_H
#define CountSort_H

#include <iostream>
#include <cstring>
#include <cstdlib>
#include <ctime>

#include <omp.h>
#include <cilk/cilk.h>
#include <cilk/cilk_api.h>

#include "CountSortTBB.hpp"

using namespace std;

class CountSort {
public:
      static void sort(int*,long);
      static                          void
sort_openmp(int*,long);
      static void sort_tbb(int*,long);
      static                          void
sort_cilk(int*,long);

      static                          void
randomize_data(int*,long);
      static void print(int*,long);
private:
      static  const  int  idealGrainSize
= 100;
};

#endif
```

```
//SINGULAR
void CountSort::sort(int* data, long
n) {
      int* temp = new int[n];
      for(long i=0;i<n;i++) {
            int count = 0;
            for(long j=0;j<n;j++) {
                  if(data[j]<data[i])
count++;
                  else
 if(data[j] == data[i] && j<i)
count++;
            }
            temp[count] = data[i];
      }
      memcpy(data,temp,n*sizeof(int));
      delete[] temp;
}
```

```
//PARALLEL
void CountSort::sort_openmp(int* data, long n)
{
      int* temp = new int[n];
      omp_set_num_threads(omp_get_num_procs());
      #pragma omp parallel for
      for(long i=0;i<n;i++) {
            long count = 0;
            #pragma omp parallel for
            for(long j=0;j<n;j++) {
                  if(data[j]<data[i]) count++;
                  else if(data[j] == data[i] &&
j<i) count++;
            }
            temp[count] = data[i];
      }
      memcpy(data,temp,n*sizeof(int));
      delete[] temp;
}
```

Dilakukan eksperimen panjang mengenai waktu aktual dari penggunaan kode tersebut dan dicatat durasi yang dibutuhkan program untuk menyelesaikan persoalan tersebut. Berikut hasil eksperimen yang telah dilakukan.

```
SINGLE with 16 data = 0 second(s)
SINGLE with 256 data = 1 second(s)
SINGLE with 4096 data = 0 second(s)
SINGLE with 65536 data = 27 second(s)
SINGLE with 1048576 data = 16221
second(s)
OPENMP with 16 data = 0 second(s)
OPENMP with 256 data = 0 second(s)
OPENMP with 4096 data = 0 second(s)
OPENMP with 65536 data = 8 second(s)
OPENMP with 1048576 data = 2729
second(s)
```

Dari pengamatan diatas proses yang memanfaatkan kemampuan komputer saat ini untuk melakukan operasi secara paralel akan meningkatkan kecepatan proses secara signifikan.

V. KESIMPULAN

Dari hasil percobaan yang didapatkan diatas dapat diambil kesimpulan bahwa algoritma non-deterministik yang sampai dengan pengembangan single core dapat

dioptimalisasi dengan penggunaan multicore. Pengembangan software juga akan lebih optimal jika diarahkan menjadi program yang didesain untuk bekerja secara paralel. Dari kesimpulan ini muncul pula apakah deterministic turing machine yang menjadi dasar pengembangan komputer sampai saat ini akan bergeser menjadi non-deterministik.

[1] Hopcroft, John E. Motwani, Rejeev. Ullman, Jeffrey D. "Introduction to Automata Theory, Languages and Computation". 2001. Pearson Education.

[2] Carnegiemellon University in Qatar. Formal Languages, Automata and Computation Slide. Spring 2011.

[3] http://www4.ncsu.edu/~uzgeorge/HamiltonCircuits7-19and22.pdfB. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished. Tanggal akses 21 Desember 2015.

[4] http://www.doc.ic.ac.uk/~mrc/Computability%20&%20Complexity/Lectures/C240Lec17.pdf. 2010. Tanggal akses 21 Desember 2015

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Desember 2015

Fitrandi Ramadhan
23515050