Pencarian Banyak Bilangan Prima Menggunakan Parallel Computing

Muhammad Daffa Dinaya (13518141)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13518141@std.stei.itb.ac.id; muhammaddaffadinaya@gmail.com

Abstrak — Salah satu pengembangan dari beberapa teknologi terbaru adalah penerapan sistem yang bekerja untuk menyelesaikan masalah yang besar. Dari masalah besar tersebut sering digunakan metode dimana masalah akan dibagi menjadi beberapa masalah yang independen dan akan diselesaikan oleh beberapa sistem secara *parallel*. Lalu bagaimana implementasi penghitungan pencarian banyak bilangan prima yang berukuran besar menggunakan sistem *parallel*?

***Keywords*—Bilangan Prima, Divide and Conquer, Kompleksitas, Parallel computing,**

I. PENDAHULUAN

Sistem secara *parallel* ini sering dilakukan apabila sistem digunakan untuk menyelesaikan permasalahan besar dan kompleks. Misal dalam penghitungan banyak bilangan dari suatu himpunan bilangan, jika melakukan penghitungan dengan cara menghitung satu-persatu bilangan secara menyeluruh akan menyebabkan sebuah bagian sistem bekerja terlalu keras dan akan memakan waktu yang cukup lama. Namun dengan penghitungan dengan sistem *parallel* sistem dapat membagi tugas untuk diproses secara independen.

Dengan pembagian tugas tersebut, sistem akan dapat melakukan penghitungan secara sekaligus dan sistem tidak akan terlalu terbebani di satu bagian saja. Dari beberapa keuntungan sistem parallel inilah yang akan dijadikan sebagai dasar pembuatan program pencarian banyak bilangan prima. Program ini akan menjalankan beberapa proses dengan cara membagi bilangan ke beberapa bagian dan proses tersebut bekerja sama untuk menyelesaikan pencarian banyak bilangan secara keseluruhan. Konsep tersebut dinamakan *parallel computing.*

Dalam implementasinya, program ini dapat sebagai gambaran tentang reduksi waktu yang digunakan program meskipun memiliki kompleksitas algoritma sama. Program ini juga dapat menjadi gambaran tentang implementasi sederhana dalam *parallel computing*.

Makalah ini hanya akan membahas penerapan secara sederhana dari *parallel computing*, pemisahan masalah sehingga dapat diterapkan *parallel computing*, dan metode penghitungan waktu yang akan digunakan. Makalah ini akan terlepas dari keberlakuan penerapan program di berbagai *environtment* tertentu dan beberapa batasan *hardware* lainnya.

II. DASAR TEORI

Berikut ini adalah beberapa dasar teori yang digunakan dalam pembuatan makalah ini

A. Bilangan Prima

Dalam pengertian definisi teori bilangan bilangan prima adalah suatu bilangan yang lebih dari 1 dengan faktor positifnya adalah dirinya sendiri dan 1. (Kenneth,2013)

Maka untuk melakukan pengecekan apakah suatu bilangan p prima atau tidak, dapat dilakukan dengan cara melakukan pembagian dengan bilangan dari 1 hingga ke p.

Namun terdapat teorema jika n adalah bilangan bukan prima, maka n memiliki pembagi bilangan prima kurang dari sama dengan √n. Maka dengan teorema ini, pengecekan hanya akan dilakukan hingga bilangan √n.

B. Kompleksitas

Kompleksitas algoritma dapat diartikan sebagai operasi yang dilakukan oleh suatu algoritma dengan suatu masukan dengan ukuran tertentu. Salah satu bentuk kompleksitas algoritma adalah kompleksitas waktu

Dalam kompeksitas waktu akan menunjukan waktu yang dibutuhkan suatu proses dijalankan oleh perangkat. Namun waktu yang diperlukan akan berbeda-beda pada setiap perangkat dengan spesifikasi berbeda. Hal ini terjadi karena setiap perangkat memiliki kemampuan berbeda dalam menangani pemrosesan sebuah operasi dasar.

Sebagai batas atas, umum digunakan notasi big-O yang menunjukkan kompleksitas waktu algoritma dalam kasus terburuk (worst case). Hal ini agar menunjukan performa minimum dari suatu algoritma sehingga menjadi standar dari algoritma tersebut

C. Divide and Conquer

Divide and Conquer secara sederhana adalah algoritma yang membagi suatu permasalah besar menjadi permasalahan kecil yang dapat diselesaikan secara terpisah. Langkah dari pemrosesan divide and conquer adalah

1. Divide : Memecah permasalahan besar menjadi kecil dan dapat diselesaikan dengan independen dan bersifat sama
2. Conquer : Memecahkan permasalahn kecil yang bersifat sama

3. Combine: Menggabungkan solusi dari pemecahan masalah yang sudah didapat

D. Parallel Computing

Parallel Computing adalah pemrosesan dengan program dimana di dalam program terdapat berbagai instruksi yang saling bekerja sama untuk menyelesaikan permasalahan (Peter, 2011). Dalam parallel computing, instruksi akan dijalankan secara bersamaan dan akan menjalankan instruksi yang dibagikan namun tetap adanya kooperasi dari program utama. Jadi *progress* dari masing-masing instruksi tetap tidak bisa berjalan secara sendiri-sendiri dan masih bergantung dengan instruksi lainnya

Dalam parallel computing terdapat istilah yang dikenal dengan threading dimana program dapat menjalankan suatu perintah dengan tugas tertentu secara independen. Hal ini dapat berguna dalam proses divide and conquer jika permasalahan yang bersifat mirip diselesaikan secara sekaligus menggunankan threading.

III. Penghitungan Banyak Bilangan Prima Dengan Single-Thread

Sebagai langkah awal pemecahan, akan dilakukan penghitungan banyak bilangan prima dari 1 hingga 5. Jika menerapkan hanya berdasar definisi, maka bentuk implementasi akan berupa fungsi GetPrime(a,b) yang akan menghasilkan banyak bilangan prima dari a hingga b, IsPrime(x) yang akan melakukan pengecekan

Prime = 0
GetPrime(1,5)
IsPrime(1) = (1 mod 1 != 0) = False
IsPrime(2) = (2 mod 2 == 0) and (2 mod 1 == 0) = True
Prime = 1
IsPrime(3) = (3 mod 3 == 0) and (3 mod 2 != 0) and (3 mod 1 == 0) = True
Prime = 2
IsPrime(4) = (4 mod 4 == 0) and (4 mod 3 != 0) and (4 mod 2 != 0) and (4 mod 1 == 0) = False
IsPrime(5) = (5 mod 5 == 0) and (5 mod 4 != 0) and (5 mod 3 != 0) and (5 mod 2 != 0) and (5 mod 1 == 0) = True
Prime = 3

Maka jumlah yang penghitungan bilangan prima yang didapat ditunjukan dengan variable *Prime* yaitu 3.

Jika menggunakan *Trial Division* hingga kurang dari sama dengan √n maka langkah pengecekan akan lebih sedikit. Ditambah lagi jika melakukan eliminasi pengecekan pada pembagian dengan bilangan 1 dan bilangan genap selain 2. Hal ini karena setiap bilangan bulat dipastikan dapat dibagi dengan 1 dan bilangan genap dipastikan bilangan komposit. Sehingga langkah penghitungan banyak bilangan prima dapat dilakukan menjadi

Prime = 0
GetPrime(1,5)
IsPrime(1) = False
IsPrime(2) = True
Prime = 1
IsPrime(3) = True
Prime = 2
IsPrime(4) = (4 mod 2 != 0) = False
IsPrime(5) = (5 mod 2 != 0) = True
Prime = 3

Dapat dilihat bahwa pengurangan langkah pengecekan cukup signifikan mengingat hanya melakukan pengecekan hingga √n. Berikut adalah potongan kode sumber prosedur GetPrime apabila diimplementasikan ke dalam bahasa C

Prosedur getPrime

```
prime = 0;
void getPrime (long start, long check)
{
	for (j = start; j <= check; j = j + 2)
	{
	numsqrt = (unsigned long)(sqrt(j));
		true = 1;
		for (i = 3; i <= numsqrt; i = i
+ 2)
		{
			if ((j % i) == 0)
			{
				true = 0;
				break;
			}
		}
		if (true)
		{
			prime++;
		}
	}
}
```

Dalam program prosedur getPrime, hanya akan melakukan pengecekan mulai dari masukan start hingga masukan check. Variable yang dirubah yaitu variable prime memiliki sifat sebagai variable global, hal ini untuk mempermudah dalam pembuatan alur program.

Penggunaan metode single thread hanya akan berfokus di bagian pengurangan langkah pengecekan. Untuk kompleksitas algoritma tersebut dapat diperoleh dengan memperhatikan langakah hanya mengambil bilangan ganjil dari start hingga check di perulangan pertama dan pengecekan keterbagian dengan bilangan ganjil dari 3 hingga akar dari bilangan yang diperiksa. Maka kompleksitas waktu dapat diperoleh menjadi

$$T(n) = n/2 * \sqrt{n-3}/2$$

Dari persamaan maka diperoleh kompleksitas melalui penghitungan sebagai berikut

$$n/2 * \sqrt{n-3}/2 = n * \sqrt{n-3}/4$$
$$n\sqrt{n-3}/4 \leq \frac{1}{4} n\sqrt{n}$$

Maka diperoleh kompleksitas dari algoritma tersebut dalam Big-O notation adalah O (n√n). Berikut ini adalah data running dari percobaan tersebut. Untuk metode penghitungan waktu lebih lanjut akan dibahas di Bab V

n	Waktu (s)	prime
100	0,000403	25
1.000	0,000503	168
10.000	0.001041	1229
100.000	0.025179	9592
1.000.000	0.461792	78498

IV. Implementasi Dalam Multi-Thread

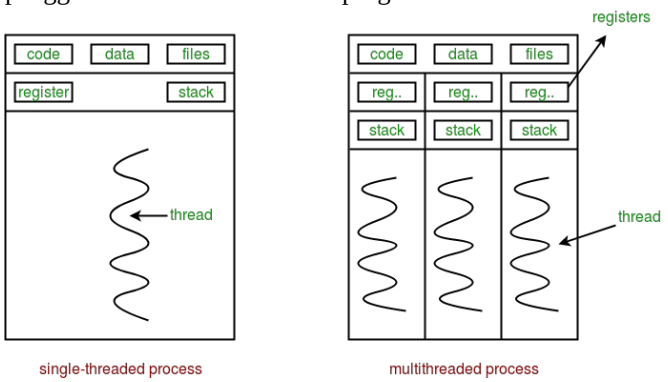
A. *Portable Operating System Interface for UNIX (POSIX)*

Implementasi program multithread, akan bergantung pada pustaka yang sudah tersedia bahasa C. Pustaka yang akan digunakan adalah pustaka POSIX Thread (Pthread). Hal ini perlu diingat bahwa POSIX hanya mendukung sistem operasi berbasis UNIX. (Peter Pacheco - An Introduction to Parallel Programming-Morgan Kaufmann (2011))

Penggunaan POSIX Thread pada dasarnya memungkinkan menjalankan beberapa instruksi secara langsung. Hal ini dilakukan dengan membuat thread yang akan berjalan secara independen. Ketika sebuah thread dijalankan maka secara otomatis program akan menjalan intruksi dari thread dan sekaligus melanjutkan instruksi dari program utama.

Dalam pembuatan thread, antara thread dengan program utama tetap akan hubungan. Hubungan tersebut terdapat pada pembagian data global yang sama. Namun selama ada data yang diambil untuk diproses di dalam area local oleh thread, tidak dapat diakses oleh pogram utama ataupun thread lain.

Untuk mempermudah pemahaman, berikut ilustrasi dari penggunaan multithread dalam program



Gambar 1. Pembagian data global dan local di dalam thread (Sumber : https://www.geeksforgeeks.org/multithreading-python-set-1/, waktu akses : 5 Desember 2019)

B. *Alur Program*

Dalam proses penghitungan bilangan prima di bab sebelumnya, variable Prime bersifat global. Namun kenyataannya apabila variable Prime diubah dengan akses melalui sebuah thread, maka thread akan menyimpan Prime yang semua global akan dimasukkan ke data sementara yang diakses secara local. Dengan demikian apabila terdapat akses yang bersamaan terhadap variable Prime yang bersamaan akan menimbulkan penolakan terhadap akses ke variable dan menjadikan penghitungan banyak bilangan prima menjadi tidak valid.

Dalam akses ke variable Prime tidak akan dilakukan proses tunggu karena akan memperlambat penghitungan. Ditambah, akses terjadi dalam urutan yang acak karena antrian akses dapat berubah-ubah tergantung pada berapa lama proses tunggu dan akan berpengaruh ke urutan antrian akses ke berikutnya.

Solusi yang cukup efektif untuk permasalahan ini adalah menggunakan variable global prime yang terpisah pada setiap thread. Semisal prime1 untuk thread 1, prime2 untuk thread2, dst. Hal ini untuk menghindari penolakan akses terhadap sebuah variable global yang diakses banyak thread.

Untuk solusi tetap mengambil dalam bentuk global karena Prime masih harus digunakan di akhir program dan dirasa mempermudah akses dari program utama. Akses dari program utama masih diperlukan untuk menggabungkan penghitungan prime dari berbagai thread. Pemilihan variable Prime sebagai parameter input/ouput dalam prosedur sebenarnya dapat dipakai, tetapi akan mempersulit mengingat pthread hanya menyediakan satu argumen untuk prosedur yang dipanggil di thread[1]. Hal ini dapat diatasi dengan tipe data struct tetapi akan mempersulit akses di dalam program utama.

Untuk alur program secara sederhana adalah sebagai berikut

1. Melakukan penentuan input bilangan n
2. Program akan diberi instruksi untuk menghitung banyak bilangan prima dari 1 hingga n
3. Pogram akan membuat k thread dan akan membagi n buah bilangan menjadi k bagian
4. Program akan memulai penghitungan waktu
5. Program akan berjalan dan melakukan penghitungan yang kemudian disimpan ke dalam k global variable Prime
6. Program akan melakukan sinkronisasi hingga semua thread selesai menjalankan instruksi
7. Program akan menjumlahkan semua nilai dari k variable Prime
8. Program akan menghentikan penghitungan waktu
9. Program akan menampilkan banyak bilangan prima dan menampilkan penghitungan waktu yang digunakan selama pemrosesan.

Beberapa fungsi dan prosedur builtin yang digunakan dalam POSIX thread[2] adalah sebagai berikut

[1]Dijelaskan dalam prosedur pthread_create

[2]Pustaka Pthread.h https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html

Deklarasi thread id

```
pthread id;
/* id akan menunjukan identitas dari sebuah
thread berupa bilangan int */
```

Prosedur phtread_create

```
pthread_create(&tid1, NULL, getPrime, (void
*) &thread1);
/* pthread_create adalah prosedur yang
digunakan untuk membentuk proses thread baru
dimana parameter yang digunakan adalah id
dari Pthread, opsi pthread_create
(menggunakan NULL jika ingin opsi default),
prosedur yang akan dijalankan dalam thread,
dan argumen yang digunakan sebagai parameter
dari prosedur yang akan dijalankan dalam
thread. Perlu diingat bahwa prosedur dan
argumen yang digunakan sebagai parameter
harus dalam tipe voia. Maka dalam prosedur
phtread_create perlu dilakukan casting ke
bentuk void.*/
```

Prosedur pthread_join

```
pthread_join(idx, NULL);
/* pthread_join adalah prosedur yang
digunakan untuk melakukan sinkronisasi thread
dengan program utama. Pthread_join akan
melakukan terminasi pembacaan instruksi
hingga thread dengan nomor id idx selesai.
Parameter dari prosedur tersebut adalah id
dari thread dan nilai yang akan menunjukan
jika thread dengan id tersebut berhenti */
```

Penggunaan pthread akan dilakukan pada langkah alur program bernomor 3 dan 6. Dimana langkah 3 akan melakukan deklarasi thread id dan pembuatan thread menggunakan pthread_create dan langkah 6 akan melakukan sinkronisasi dengan pthread_join.

Perlu diperhatikan bahwa pembagian n buah bilangan pada langkah 3 dapat menentukan waktu yang diperlukan untuk pemrosesan. Hal ini disebabkan distribusi beban yang tidak merata dalam bilangan. Bilangan di bagian awal akan memiliki penghitungan relatif lebih cepat dari pada bilangan di bagian akhir. Pada bagian awal pengecekan bilangan hanya akan melakukan langkah yang lebih sedikit karena bilangan yang relatif kecil. Sedangkan bagian akhir dari n bilangan akan relatif besar dan akan mengalami banyak langkah pengecekan.

Untuk solusi pembagian bilangan akan dipilih menggunakan barisan dengan beda selisih. Gambaran dari beda selisih ini adalah

Thread	Barisan Beda Selisih
1	3, 11, 19, 27, ...
2	5, 13, 21, 29, ...
3	7, 15, 23, 31, ...
4	9, 17, 25, 33, ...

Perlu diperhatikan bahwa pembagian dengan barisan beda selisih memiliki selisih yang berbeda untuk setiap jumlah thread yang digunakan. Selisih dari barisan akan menggunakan 2n dengan n adalah banyak thread yang digunakan.

C. Kompleksitas Program dengan Multithread

Kompleksitas waktu dapat diperoleh dengan k thread menjadi

$$T(n)=\frac{(n/2*\sqrt{n-3}/2)}{k}$$

Dari persamaan maka diperoleh kompleksitas melalui penghitungan sebagai berikut

$$\frac{(n/2*\sqrt{n-3}/2)}{k}=\frac{n*\sqrt{n-3}}{4k}$$

$$n\frac{\sqrt{n-3}}{4k}\leq\frac{1}{4k}n\sqrt{n}$$

Maka diperoleh kompleksitas dari algoritma tersebut dalam Big-O notation adalah O (n√n)

V. METODE PENGUKURAN DAN ANALISIS WAKTU

A. Metode

Pengukuran dilakukan dengan mengukur waktu lama program berjalan menggunakan pustaka sys/time. Pemilihan psutaka sys/time karena terdapat opsi pengukuran waktu monotonic. Pengukuran waktu monotonic adalah pengukuran waktu yang tidak bergantung pada penghitungan waktu sistem dan akan terus bertambah secara linear dengan kontinu (Rober2007)). Hal ini agar memperoleh pengukuran waktu yang presisi.

Potongan kode sumber untuk mengukur waktu proses adalah sebagai berikut

Monotonic Time

```
#include <sys/time.h>
struct timespec start_time, end_time;
clock_gettime(CLOCK_MONOTONIC, &start_time);
clock_gettime(CLOCK_MONOTONIC, &end_time);
       double cpu_time_used;
```

```
/* Measured Process */
      cpu_time_used = (end_time.tv_sec -
start_time.tv_sec) * 1e9;
    cpu_time_used = (cpu_time_used +
(end_time.tv_nsec - start_time.tv_nsec)) *
1e-9;
```

Perlu diperhatikan bahwa variable cpu_time_used kedua adalah untuk menambkan waktu dalam nano detik[3].

B. Eksperimen

Berikut ini adalah spesifikasi cpu yang digunakan untuk pemrosesan program. Penggunaan cpu hanya sebagai perbandingan dan pembahasan tentang *environtment* maupun *hardware* lainnya tidak akan dibahas seperti yang sudah dijelaskan.

Informasi mengenai cpu yang digunakan dilakukan dengan memeriksa cpu melalui *command* "lscpu" melalui terminal

Perangkat A

Linux	Ubuntu
Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4
On-line CPU(s) list:	0-3
Thread(s) per core:	2
Core(s) per socket:	2
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	61
Model name:	Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz
Stepping:	4
CPU MHz:	1860.668
CPU max MHz:	2000,0000
CPU min MHz:	500,0000
BogoMIPS:	4190.07
Virtalization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	3072K
NUMA node0 CPU(s):	0-3

Perangkat B

Linux	Ubuntu
Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4
On-line CPU(s) list:	0-3
Thread(s) per core:	2
Core(s) per socket:	2
Socket(s):	1
NUMA node(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	142
Model name:	Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz
Stepping:	9
CPU MHz:	500.022
CPU max MHz:	3500,0000
CPU min MHz:	400,0000
BogoMIPS:	5808.07
Virtalization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	4096K
NUMA node0 CPU(s):	0-3

Perolehan waktu rata-rata dari 5 kali eksekusi program pada single, double, dan quad thread adalah sebagai berikut

Perangkat	Single (s)	Double (s)	Quad (s)
A	11,2455738	5,9009361280	4,0711348834
B	6,6120372	3,3412075008	1,9832380356

VI. KESIMPULAN

Dari perolehan penghitungan waktu, penambahan thread tidak dapat mengurangi kompleksitas program karena hanya memangkas pemrosesan program secara konstan. Penambahan thread berdampak relatif sama dengan pengurangan waktu proses. Namun, terdapat pengecualian terhadap perangkat A di

[3]Monotonic clock dokumentasi dalam pustaka time.h fungsi clock_gettime https://pubs.opengroup.org/onlinepubs/009695399/basedefs/time.h.html

program dengan quad thread. Jadi dapat disimpulkan bahwa implementasi program di atas dapat bekerja secara parallel karena mencapai tujuan yang didapat dengan penerapan konsep *parallel*

VII. LAMPIRAN

Untuk melihat program secara utuh dapat melihat dokumentasi yang sudah dibuat penulis dan menjadi dasar penulisan makalah

https://github.com/mdaffad/multithread

REFERENCES

[1] Robert L., “Linux system programming”, California:O'Reilly Media , 2007.

[2] Kenneth H. R., “Discrete Mathematics and Its Application”, New-York:McGraw-Hill , 2013.

[3] Peter P., “An Introduction to Parallel Programming”, Burlington:Morgan Kaufmann,2011.

[4] Luqman A. S., “Penyelesaian Barisan Rekursif dengan Kompleksitas Logaritmik Menggunakan Pemangkatan Matriks”, 2014.

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, 6 Desember 2019

Muhammad Daffa Dinaya (13518141)