

Virtual Machine dan Parallel Processing in R

Sebuah Learning Forum

Ikang Fadhli

Market Research @nutrifood

18 January 2022

- 1 MUKADIMAH
- 2 *VIRTUAL MACHINE*
- 3 *PARALLEL PROCESSING*

Section 1

MUKADIMAH

Learning Forum Hari Ini

Pada hari ini, kita akan membahas dan berdiskusi tentang beberapa topik, yakni:

① *Virtual Machine*

- *Linux OS*
- *Google Cloud Virtual Machine*

② *Parallel Processing*

- *Serial Processing*
- *Embarassingly Parallel*
- *Parallel yang Sebenarnya*

Section 2

VIRTUAL MACHINE

Apa itu *Virtual Machine*?

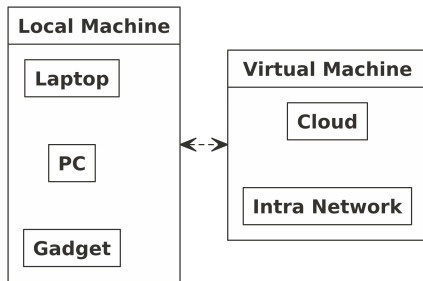
Pada konsep *computing*, *virtual machine* adalah *virtualization* atau *emulation* dari suatu sistem komputer.

Virtual machine berbasis suatu arsitektur komputer tertentu dan berfungsi sama halnya dengan komputer pada biasanya.

Salah satu *operating system* yang paling sering digunakan pada *virtual machine* adalah LINUX OS.

Android merupakan salah satu *mobile OS* yang berbasis Linux.

Apa itu *Virtual Machine*?



Pembahasan pada **LEFO** kali ini dibatasi untuk *cloud VM*. Istilah lain yang sering digunakan untuk menyebut *VM* di *cloud* adalah *virtual private server* (VPS).

Apa itu LINUX OS?

Linux adalah *operating system* berbasis UNIX (sama halnya dengan MacOS) yang bersifat *open source*. Dibuat oleh seorang *programmer* bernama **Linus Torvalds** pada 1991. Sampai saat ini, OS Linux merupakan OS yang populer digunakan di berbagai *server* dan *high performance computer* di dunia.

Ada banyak *distro* Linux yang berkembang (dengan basis komunitas yang kuat) antara lain:

- Ubuntu,
- Debian,
- Kali,
- Arch,
- Fedora.

Setiap distro memiliki spesialisasi tertentu sesuai dengan komunitas pengembang dan tujuannya.

Distro yang paling sering digunakan pada *server* dan *HPC* adalah Debian atau Ubuntu.

Fun Fact

Linus Torvalds juga salah seorang *creator* dari Git.

Apa itu LINUX OS?

Linux memiliki dua versi, yakni:

- 1 *Command line interface.*
- 2 *Graphical user interface.*

Kita bisa memilih versi mana yang hendak kita gunakan di *virtual machine*. Tapi versi **CLI** lebih disukai karena lebih ringan dan bisa memaksimalkan kemampuan *processors* yang dimiliki.

Apa itu LINUX OS?

Linux sangat ringan, bahkan bisa dijalankan dengan **USB drive** tanpa harus di-*install* terlebih dahulu di laptop/komputer.

Selain itu, Linux bisa di-*install* di berbagai komputer dan *gadget* apapun. Misal: HP / tablet Android dan laptop Windows.

Di *gadget* Android, kita bisa mencicipi Linux CLI dan GUI dengan meng-*install apps* bernama userLand. Pada laptop Windows, kita bisa meng-*install* berbagai *virtual box* (misalkan MobaXterm).

Google Virtual Machine

Berbekal *credit* yang kita dapatkan saat mengaktifkan layanan *Google Cloud Service*, kita bisa menyewa *virtual machine* sesuai dengan kebutuhan.

Bagaimana caranya?

Google Virtual Machine

DEMO

UserLand di Android

DEMO

Diskusi

Any comments?

Section 3

PARALLEL PROCESSING

Apa Tujuannya *Parallel Processing*?

Manusia itu terobsesi dengan hal-hal yang bisa dilakukan dengan cepat. Apalagi jika sumber daya komputasi yang dimiliki sangat mumpuni.

Jika suatu komputasi bisa dilakukan lebih cepat, kenapa tidak?

Lebih Cepat, Lebih Baik!



Sekilas tentang *Runtime*

Runtime suatu proses komputasi bergantung pada dua hal:

- ① Kompleksitas algoritma (termasuk bahasa pemrograman yang dipilih).
- ② Kapasitas *hardware* (termasuk *memory allocation*).

Oleh karena itu, *runtime* baik *serial processing* dan *parallel processing* sangat berpengaruh pada kedua faktor di atas.

Serial Processing vs Parallel Processing

Serial Processing

Adalah proses komputasi yang bersifat sekuensial. Misal suatu proses komputasi memiliki n proses. Proses ke i tidak bisa dilakukan hingga proses ke $i - 1$ selesai ($i \leq n$).

Parallel Processing

Adalah proses komputasi dimana semua proses yang terlibat dijalankan secara bersamaan. Misalkan suatu proses komputasi memiliki n proses dijalankan bersamaan. Proses ke i bisa dijalankan tanpa menunggu proses yang lain.

Analogi



Menurut kalian proses perakitan satu mobil ini berlangsung secara serial atau parallel?

Analogi

Suatu ketika, saya diminta tolong untuk memberikan informasi kepada 8 orang via *messaging*.

Apa harus saya lakukan jika saya menggunakan:

- 1 *Serial Processing*
- 2 *Parallel Processing*

Analogi

Dalam *parallel processing* ada yang disebut dengan *embarassingly parallel*. Apakah se-memalukannya itu?

Contoh I

Saya membuat grup WA berisi 8 orang tersebut. Lalu menginformasikannya melalui grup tersebut.

Contoh II

Saya meminta tolong orang lain sehingga menggunakan 8 buah *gadgets* untuk mengirimkan informasi tersebut kepada 8 orang secara bersamaan.

Pertanyaannya

Antara contoh I dan II, mana yang memalukan?

Apakah *Serial Processing* Buruk?

The screenshot shows a Linux terminal window with the following content:

```

1  [ 0.7%] 5 [ 0.0%]
2  [ 100.0%] 6 [ 0.7%]
3  [ 1.3%] 7 [ 1.9%]
4  [ 0.0%] 8 [ 2.8%]
Mem [ 6.16G/15.2G] Tasks: 167, 860 thr; 2 running
Sup [ 0K/3.73G] Load average: 1.62 0.83 0.61
Uptime: 06:17:57

PID USER      PRI  NI  VIRT   RES   SHR  S CPU% MEM%   time+   Command
27069 ikannx10i 20   0  635M 52172 40284 S  1.3 0.3 0:01.99 /usr/libexec/gn
8447 ikannx10i 20   0  17.9G 616M 159M S  0.7 4.0 16:28.04 /snap/ranbox/18
29184 ikannx10i 20   0 11288 4756 3384 R  0.7 0.0 0:00.15 htop
8565 ikannx10i 20   0 4798M 273M 112M S  0.7 1.8 6:35.57 /snap/ranbox/18
4045 ikannx10i 20   0 16.4G 317M 179M S  0.7 2.0 10:36.30 /opt/google/chr
6222 ikannx10i 20   0 12.8G 429M 170M S  0.7 2.8 6:03.33 /snap/ranbox/18
8566 ikannx10i 20   0 4798M 273M 112M S  0.7 1.8 2:55.79 /snap/ranbox/18
8440 ikannx10i 20   0 17.9G 616M 139M S  0.7 4.0 5:03.50 /snap/ranbox/18
6231 ikannx10i 20   0 69.0G 364M 150M S  0.7 2.3 1:57:41 /snap/ranbox/18
6319 ikannx10i 20   0 13.9G 555M 186M S  0.0 3.6 10:56.01 /snap/ranbox/18
29163 ikannx10i 20   0 28.4G 117M 85940 S  0.0 0.8 0:00.65 /opt/google/chr
29190 ikannx10i 20   0 28.4G 117M 85940 S  0.0 0.8 0:00.04 /opt/google/chr

```

On the right side of the terminal, there is a script being executed:

```

> rm(list=ls())
> # initial condition
> nx = 10^6
> fx = function(x){4 * sqrt(1 - x^2)}
> # serial
> hitung_pi_serial = function(n){
+   mulai = Sys.time()
+   sum = 0
+   for(i in 1:n){
+     xi = runif(1)
+     sum = sum + fx(xi)
+     .... [TRUNCATED]
+   }
+ }
> print("Hasil Menggunakan Serial Processing!")
[1] "Hasil Menggunakan Serial Processing:"
> hitung_pi_serial(nx)

```

Sejatinnya saat **R** atau **Python** melakukan *process* terhadap suatu algoritma, **mereka hanya memakai 1 core komputer kita saja.**

Ide Dasar *Parallel Processing*

Jika suatu proses komputasi bisa dilakukan selama X detik pada *single processor*, maka jika kita memiliki n buah *processors* prosesnya seharusnya bisa dihemat menjadi $\frac{X}{n}$ detik.

Setuju? Logis gak?

Tapi...

Percepatan *runtime* ini tidak berlaku *linear* seperti di atas. Tapi yang jelas memang lebih cepat dibanding *serial processing*.

Lantas seberapa cepat?

Berbicara Mengenai *Parallel Processing*

Dahulu kala saat berbicara mengenal *parallel processing*, istilah ini hanya digunakan saat seseorang melakukan komputasi menggunakan *high performance computer* (HPC). HPC merupakan “super komputer” yang dibuat dari beberapa komputer (multi *processors*) yang dibuat sedemikian rupa (menggunakan arsitektur jaringan tertentu).

Namun demikian, istilah ini sekarang bisa untuk segala jenis komputer. Bahkan *gadget* Android yang kita pakai sekarang sudah *multicores* sehingga memungkinkan untuk melakukan *parallel processing*.

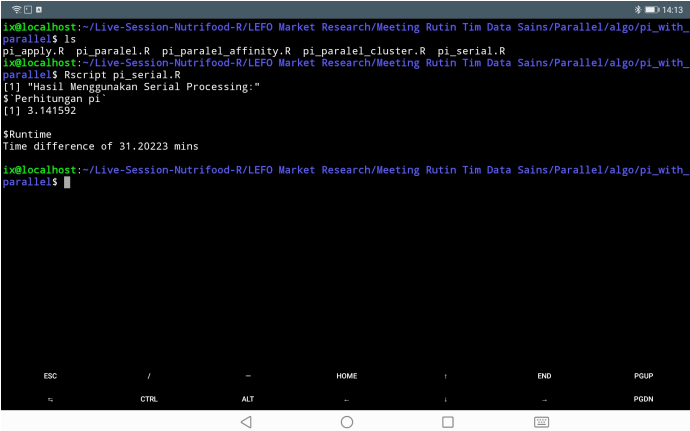
Contoh

Deteksi berapa banyak *cores* di Huawei T10s.

```
ix@localhost:~/ikanx101.github.io/_posts/matematika ITB/server/post3$ lscpu
Architecture:      aarch64
Byte Order:        Little Endian
CPU(s):            8
On-line CPU(s) list: 0-7
Thread(s) per core: 1
Core(s) per socket: 4
Socket(s):         2
Vendor ID:         ARM
Model:             4
Model name:        Cortex-A53
Stepping:          r0p4
CPU max MHz:       1997.0000
CPU min MHz:       480.0000
BogoMIPS:          3.84
Flags:             fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
```

Contoh

Runtime untuk menjalankan `pi.serial.R` di Huawei T10s:



```
ix@localhost:~/Live-Session-Nutrifood-R/LEFO Market Research/Meeting Rutin Tim Data Sains/Parallel/algo/pi_with_parallel$ ls
pi_apply.R pi_parallel.R pi_parallel_affinity.R pi_parallel_cluster.R pi_serial.R
ix@localhost:~/Live-Session-Nutrifood-R/LEFO Market Research/Meeting Rutin Tim Data Sains/Parallel/algo/pi_with_parallel$ Rscript pi_serial.R
[1] "Hasil Menggunakan Serial Processing:"
$ Perhitungan pi
[1] 3.141592

$Runtime
Time difference of 31.20223 mins

ix@localhost:~/Live-Session-Nutrifood-R/LEFO Market Research/Meeting Rutin Tim Data Sains/Parallel/algo/pi_with_parallel$
```

Kita Mulai Bedah *Parallel Processing*

Runtime suatu proses komputasi bergantung pada dua hal:

- 1 Kompleksitas algoritma (termasuk bahasa pemrograman yang dipilih).
- 2 Kapasitas *hardware* (termasuk *memory allocation*).

Kapasitas *Hardware*

Salah satu faktor yang berpengaruh adalah *clock* dari *processor*. Maka semakin banyak *processors* maka diharapkan prosesnya semakin cepat. *Memory allocation* juga (konon) berpengaruh.

Pada *Python*, *parallelism* yang menggunakan konsep *message parsing interface* diperlukan *middleware* untuk menghubungkan *Python* dan *processors*. Kita bisa gunakan *middleware* bernama **openMPI**.

Kompleksitas Algoritma

Perhitungan kompleksitas waktu $T(n)$ diukur dari tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari *input* n .

Sebagai contoh, suatu algoritma yang digunakan untuk menghitung rata-rata dari suatu data $\{1, 2, \dots, n\}$ sebagai berikut:

```
sum = 0
for i in 1 to n:
    sum = sum + i
avg = sum / n
```

Memiliki kompleksitas waktu $T(n) = n$. Dihitung dari operasi mendasar di dalamnya yakni $\text{sum} = \text{sum} + i$ yang diulang sebanyak n kali.

Kompleksitas Algoritma

Akibatnya waktu komputasi yang diperlukan menjadi lebih banyak karena **tahapan yang dilalui juga banyak**.

Oleh karena itu, salah satu hal yang bisa dilakukan dalam *parallel processing* adalah mengubah paradigma membuat algoritma, dari sekuensial ke bentuk *parallel*.

Mengubah Paradigma

Bahasa pemrograman modern saat ini sudah memungkinkan komputer untuk memproses suatu *array* secara sekaligus. Prinsip sederhana ini yang bisa digunakan dalam membuat algoritma *embarassingly parallel processing*.

Contohnya di **R** adalah penggunaan:

- 1 `apply()`
- 2 `sapply()`
- 3 `lapply()`

Penggunaan Keluarga `apply()`

Penggunaan keluarga `apply()` membuat proses jauh lebih cepat karena data diproses secara *array*. Namun *memory* yang dibutuhkan biasanya akan semakin besar.

Namun, perlu dipahami bahwa yang terjadi adalah **kita tetap menggunakan 1 core saja**.

Runtime dengan apply()

The screenshot shows an RStudio interface with a live session. The main window displays a terminal window with the following output:

```

1 3.9% 5 4.5%
2 98.1% 6 2.0%
3 3.3% 7 5.2%
4 6.5% 8 4.6%
Mem: 6.67G/15.2G Tasks: 163, 793 thr: 2 running
Swap: 0K/3.73G Load average: 1.02 0.87 0.87
Uptime: 06:51:07

```

Below the terminal, a table of processes is displayed:

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
9205	ikanx_ser	20	0	1567M	702M	63608	R	96.2	4.5	5:15.61	/usr/lib/rstudio
4335	ikanx101g	20	0	1080	455M	120M	S	9.7	2.9	17:03.11	/opt/google/chr
1812	ikanx101g	20	0	5024M	300M	104M	S	6.5	1.9	10:45.04	/usr/bin/gnome-
4092	ikanx101g	20	0	16.66	160M	114M	S	5.2	1.0	10:52.62	/opt/google/chr
1661	ikanx101g	20	0	1088M	120M	81308	S	3.2	0.8	9:56.80	/usr/lib/xorg/X
4045	ikanx101g	20	0	16.86	328M	186M	S	1.9	2.1	11:49.22	/opt/google/chr
4341	ikanx101g	20	0	1086	455M	120M	S	1.9	2.9	2:57.60	/opt/google/chr
8565	ikanx101g	20	0	4798M	274M	112M	S	1.3	1.8	7:34.85	/snap/rambox/18
8447	ikanx101g	20	0	21.06	618M	139M	S	1.3	4.0	17:34.43	/snap/rambox/18
27069	ikanx101g	20	0	825M	52312	40284	S	1.3	0.3	0:07.31	/usr/libexec/gn
31529	ikanx101g	20	0	11248	4416	3236	R	1.3	0.0	0:00.20	htop
4339	ikanx101g	20	0	1080	455M	120M	S	1.3	2.9	1:16.93	/opt/google/chr
4134	ikanx101g	20	0	16.86	160M	114M	S	1.3	1.0	3:33.59	/opt/google/chr

The right-hand pane shows the Environment tab with a list of objects and the Console tab with the following R code:

```

> rm(list=ls())
> # initial condition
> nx = 10^8
> fx = function(x){4 * sqrt(1 - x^2)}
> # paralel
> hitung_pi = function(n){
+   mulai = Sys.time()
+   xi = runif(n)
+   fx1 = fx(xi)
+   pi_numerik = mean(fx1)
+   waktu = Sys.time() - n .... [TRUNCATED]
> print("Hasil Menggunakan apply:")
[1] "Hasil Menggunakan apply:"
> lapply(nx, hitung_pi)

```

Bagaimana Agar Memaksimalkan Semua Cores?

Setidaknya ada berbagai cara menuliskan algoritma dan *libraries* yang bisa melakukan *parallel processing*. Salah satu *library* bawaan **R** yang bisa kita pakai adalah `library(parallel)`.

The screenshot shows an RStudio session on a Linux system. The script in the editor defines a function `hitung_pi_parallel` that uses `parallel::mclapply` to calculate pi in parallel across multiple cores. The terminal window shows the output of the script, including the number of tasks and threads running, and a table of system resource usage for each process.

```

# RStudio Script: parallel_pi.R
1 rm(list=ls())
2 set.seed(2022)
3
4 # initial condition
5 nx = 10^8
6
7 # fungsi
8 fx = function(x){1 + sqrt(1 - x^2)}
9
10 # parallel
11 hitung_pi_parallel = function(n){
12   xl = runif(x)
13   fxl = fx(xl)
14   p1_namerik = mean(fxl)
15   return(p1_namerik)
16 }
17
18 # parallel
19 library(parallel)
20 numCores = detectCores()
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Terminal Output:

```

> pi = mclapply(list(n_pecah_n_peca,
+   h_n_pecah_n_peca,
+   h_n_pecah_n_peca),
+   hitung_pi_parallel, mc.pro
+   sched .... [TRUNCATED]
> pi = mean(unlist(pi))
> waktu = Sys.time() - mulai
> print(pi)
[1] 3.14159
> print(waktu)
Time difference of 1.091897 secs

```

System Resource Usage Table:

PID	USER	PR	NI	VIRT	RES	SHR	S	CPUS	MEM%	TIME+	TIME-
1	kanan	0	0	0	0	0	0	0	0	0	0
2	kanan	0	0	0	0	0	0	0	0	0	0
3	kanan	0	0	0	0	0	0	0	0	0	0
4	kanan	0	0	0	0	0	0	0	0	0	0
5	kanan	0	0	0	0	0	0	0	0	0	0
6	kanan	0	0	0	0	0	0	0	0	0	0
7	kanan	0	0	0	0	0	0	0	0	0	0
8	kanan	0	0	0	0	0	0	0	0	0	0
9	kanan	0	0	0	0	0	0	0	0	0	0
10	kanan	0	0	0	0	0	0	0	0	0	0
11	kanan	0	0	0	0	0	0	0	0	0	0
12	kanan	0	0	0	0	0	0	0	0	0	0
13	kanan	0	0	0	0	0	0	0	0	0	0
14	kanan	0	0	0	0	0	0	0	0	0	0
15	kanan	0	0	0	0	0	0	0	0	0	0
16	kanan	0	0	0	0	0	0	0	0	0	0
17	kanan	0	0	0	0	0	0	0	0	0	0
18	kanan	0	0	0	0	0	0	0	0	0	0
19	kanan	0	0	0	0	0	0	0	0	0	0
20	kanan	0	0	0	0	0	0	0	0	0	0
21	kanan	0	0	0	0	0	0	0	0	0	0
22	kanan	0	0	0	0	0	0	0	0	0	0
23	kanan	0	0	0	0	0	0	0	0	0	0
24	kanan	0	0	0	0	0	0	0	0	0	0
25	kanan	0	0	0	0	0	0	0	0	0	0
26	kanan	0	0	0	0	0	0	0	0	0	0
27	kanan	0	0	0	0	0	0	0	0	0	0
28	kanan	0	0	0	0	0	0	0	0	0	0
29	kanan	0	0	0	0	0	0	0	0	0	0
30	kanan	0	0	0	0	0	0	0	0	0	0
31	kanan	0	0	0	0	0	0	0	0	0	0
32	kanan	0	0	0	0	0	0	0	0	0	0
33	kanan	0	0	0	0	0	0	0	0	0	0
34	kanan	0	0	0	0	0	0	0	0	0	0
35	kanan	0	0	0	0	0	0	0	0	0	0
36	kanan	0	0	0	0	0	0	0	0	0	0
37	kanan	0	0	0	0	0	0	0	0	0	0
38	kanan	0	0	0	0	0	0	0	0	0	0
39	kanan	0	0	0	0	0	0	0	0	0	0
40	kanan	0	0	0	0	0	0	0	0	0	0
41	kanan	0	0	0	0	0	0	0	0	0	0
42	kanan	0	0	0	0	0	0	0	0	0	0
43	kanan	0	0	0	0	0	0	0	0	0	0
44	kanan	0	0	0	0	0	0	0	0	0	0
45	kanan	0	0	0	0	0	0	0	0	0	0
46	kanan	0	0	0	0	0	0	0	0	0	0
47	kanan	0	0	0	0	0	0	0	0	0	0
48	kanan	0	0	0	0	0	0	0	0	0	0
49	kanan	0	0	0	0	0	0	0	0	0	0
50	kanan	0	0	0	0	0	0	0	0	0	0
51	kanan	0	0	0	0	0	0	0	0	0	0
52	kanan	0	0	0	0	0	0	0	0	0	0
53	kanan	0	0	0	0	0	0	0	0	0	0
54	kanan	0	0	0	0	0	0	0	0	0	0
55	kanan	0	0	0	0	0	0	0	0	0	0
56	kanan	0	0	0	0	0	0	0	0	0	0
57	kanan	0	0	0	0	0	0	0	0	0	0
58	kanan	0	0	0	0	0	0	0	0	0	0
59	kanan	0	0	0	0	0	0	0	0	0	0
60	kanan	0	0	0	0	0	0	0	0	0	0
61	kanan	0	0	0	0	0	0	0	0	0	0
62	kanan	0	0	0	0	0	0	0	0	0	0
63	kanan	0	0	0	0	0	0	0	0	0	0
64	kanan	0	0	0	0	0	0	0	0	0	0
65	kanan	0	0	0	0	0	0	0	0	0	0
66	kanan	0	0	0	0	0	0	0	0	0	0
67	kanan	0	0	0	0	0	0	0	0	0	0
68	kanan	0	0	0	0	0	0	0	0	0	0
69	kanan	0	0	0	0	0	0	0	0	0	0
70	kanan	0	0	0	0	0	0	0	0	0	0
71	kanan	0	0	0	0	0	0	0	0	0	0
72	kanan	0	0	0	0	0	0	0	0	0	0
73	kanan	0	0	0	0	0	0	0	0	0	0
74	kanan	0	0	0	0	0	0	0	0	0	0
75	kanan	0	0	0	0	0	0	0	0	0	0
76	kanan	0	0	0	0	0	0	0	0	0	0
77	kanan	0	0	0	0	0	0	0	0	0	0
78	kanan	0	0	0	0	0	0	0	0	0	0
79	kanan	0	0	0	0	0	0	0	0	0	0
80	kanan	0	0	0	0	0	0	0	0	0	0
81	kanan	0	0	0	0	0	0	0	0	0	0
82	kanan	0	0	0	0	0	0	0	0	0	0
83	kanan	0	0	0	0	0	0	0	0	0	0
84	kanan	0	0	0	0	0	0	0	0	0	0
85	kanan	0	0	0	0	0	0	0	0	0	0
86	kanan	0	0	0	0	0	0	0	0	0	0
87	kanan	0	0	0	0	0	0	0	0	0	0
88	kanan	0	0	0	0	0	0	0	0	0	0
89	kanan	0	0	0	0	0	0	0	0	0	0
90	kanan	0	0	0	0	0	0	0	0	0	0
91	kanan	0	0	0	0	0	0	0	0	0	0
92	kanan	0	0	0	0	0	0	0	0	0	0
93	kanan	0	0	0	0	0	0	0	0	0	0
94	kanan	0	0	0	0	0	0	0	0	0	0
95	kanan	0	0	0	0	0	0	0	0	0	0
96	kanan	0	0	0	0	0	0	0	0	0	0
97	kanan	0	0	0	0	0	0	0	0	0	0
98	kanan	0	0	0	0	0	0	0	0	0	0
99	kanan	0	0	0	0	0	0	0	0	0	0
100	kanan	0	0	0	0	0	0	0	0	0	0

Setidaknya ada 3 Cara di parallel

Kita bisa memanfaatkan *function* `mclapply()` dengan memanfaatkan beberapa parameter berikut:

- ➊ Menggunakan *all cores*.
- ➋ Menggunakan *all cores* dan meng-*assign* setiap *cores* dengan tugas-tugas tertentu.

Kita bisa membuat *cluster nodes* dan menggunakan perintah `parSapply()` dan `parLapply()`

DEMO

Pada demo ini, kita akan menghitung nilai π dengan cara melakukan integral dari fungsi berikut:

$$\int_0^1 4(1 - x^2)dx$$

Secara numerik, berarti kita mencari luas dari kurva berikut:

