

JARINGAN DAN PENGOLAHAN DATA PARALEL

LAPORAN PRAKTIKUM

Mohammad Rizka Fadhli

Ikang

20921004@mahasiswa.itb.ac.id

31 December 2021

Contents

1	<i>INTRODUCTION</i>	5
1.1	Definisi	5
1.2	Perbedaan <i>Serial Processing</i> dan <i>Parallel Processing</i>	5
1.3	Cara Kerja <i>Parallel Processing</i>	7
1.3.1	SISD	8
1.3.2	SIMD	8
1.3.3	MISD	8
1.3.4	MIMD	9
1.4	<i>SERVER</i> YANG DIGUNAKAN	9
2	<i>METHOD</i>	9
2.1	MPI	12
3	<i>RESULT AND DISCUSSION</i>	12
3.1	Soal I	12
3.2	Soal II	12
3.3	Soal III	12
4	<i>CONCLUSION</i>	12
	<i>REFERENCES</i>	13

List of Figures

1	Ilustrasi Perbedaan Serial dan Parallel Processing	7
2	Ilustrasi SISD	8
3	Ilustrasi SIMD	8
4	Spesifikasi Server yang Digunakan	10
5	Tampilan Awal Setelah Login ssh	10
6	lscpu dari Server	11
7	htop dari Server	11
8	Versi MPI yang Digunakan	12

List of Tables

1 INTRODUCTION

1.1 Definisi

Parallel processing adalah metode komputasi untuk menggunakan dua atau lebih *processors* untuk menjalankan beberapa tugas secara terpisah atau secara keseluruhan. Setiap komputer yang memiliki lebih dari satu *CPUs* atau memiliki *processor multi cores* bisa melakukan *parallel processing*.¹

1.2 Perbedaan *Serial Processing* dan *Parallel Processing*

Perbedaan mendasar dari *serial processing* dan *parallel processing* adalah dari segi bagaimana komputer melakukan proses komputasi. *Serial processing* berarti komputer melakukan tugasnya secara sekuensial (berurutan) menggunakan satu *processor*. Akibatnya adalah saat melakukan suatu proses yang kompleks, *runtime* yang diperlukan lebih lama karena *processor* harus memproses data satu-persatu.

Berbeda halnya dengan *parallel processing*. Tugas yang dilakukan komputer didistribusikan kepada sejumlah *processors* untuk diolah secara bersamaan. Konsekuensinya adalah *runtime* komputasi lebih singkat. Namun perlu diperhatikan dengan seksama bahwa tidak semua tugas bisa kita buat paralelisasinya dan cara kita menulis algoritma atau *coding* harus disesuaikan.

Kenapa tidak semua tugas bisa diparalelisasi?

Beberapa tugas sekuensial yang tidak bisa dihindari tidak bisa diparalelisasi.

Sebagai contoh:

1. *Looping* yang prosesnya tidak saling bergantung bisa diparalelisasi. Misalkan ada suatu fungsi untuk menghitung suatu *array* bisa diparalelisasi dengan cara memecah *array* tersebut untuk diproses bersamaan di beberapa *processors*.
2. *Looping* yang prosesnya saling bergantung tidak bisa diparalelisasi. Misalkan suatu *looping* ke i nilainya bergantung pada proses *looping* ke $i - 1$.

¹<https://searchdatacenter.techtarget.com/definition/parallel-processing>

Berikut adalah ilustrasi perbedaan serial dan *parallel processing*:

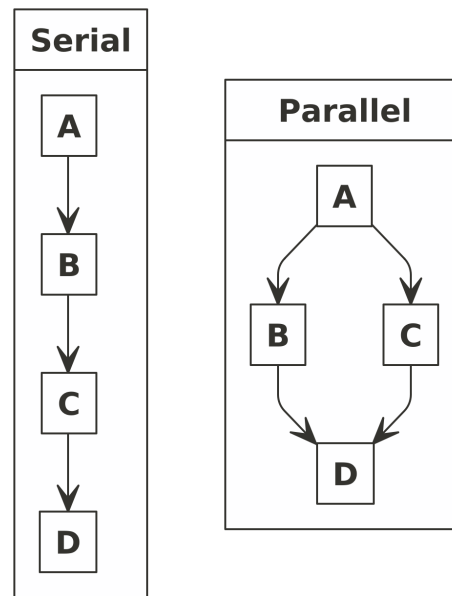


Figure 1: Ilustrasi Perbedaan Serial dan Parallel Processing

1.3 Cara Kerja *Parallel Processing*

Untuk melakukan *parallel processing*, dibutuhkan *hardware* dan *software* yang mendukung hal tersebut. Secara *hardware* dibutuhkan komputer dengan *multiple cores processors* atau dibutuhkan beberapa komputer yang digabung menjadi satu kesatuan. Secara *software* dibutuhkan tidak hanya `Python` tapi juga *middleware* bernama `open MPI`. Bagian *hardware* dan *software* ini akan dibahas pada bagian `METHOD`.

Pada sistem *parallel processing* terdiri dari beberapa unit *processors* dan beberapa unit *memory*. Ada dua teknik berbeda yang digunakan untuk mengakses data di unit *memory*, yaitu: *shared memory address* dan *message passing*.

Berdasarkan cara mengorganisasikan memori ini komputer bisa dibedakan menjadi *shared memory parallel machine* dan *distributed memory parallel machine*.

Ada empat model komputasi yang dikenal dalam taksonomi Flynn, yaitu:

1. **SISD** (*Single Instruction, Single Data*)
2. **SIMD** (*Single Instruction, Multiple Data*)

3. **MISD** (*Multiple Instruction, Single Data*)
4. **MIMD** (*Multiple Instruction, Multiple Data*)

1.3.1 SISD

Komputer ini adalah tipikal komputer konvensional yang hanya memiliki satu *processor* dan satu instruksi yang dieksekusi secara serial. Komputer jenis ini tidak bisa melakukan *parallel processing*.

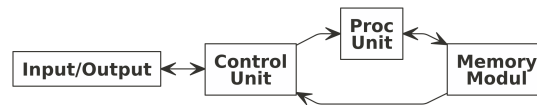


Figure 2: Ilustrasi SISD

1.3.2 SIMD

Komputer ini memiliki lebih dari satu *processor* tapi hanya mengeksekusi satu instruksi secara paralel pada data yang berbeda pada level *lock-step*. Contohnya adalah komputer vektor.

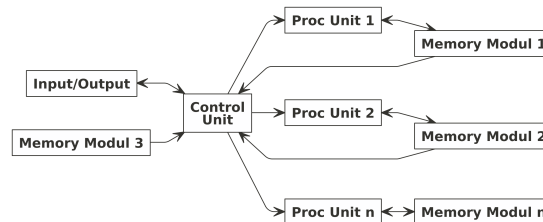


Figure 3: Ilustrasi SIMD

1.3.3 MISD

Komputer jenis ini belum diciptakan karena secara arsitekturnya tidak mudah dipahami. Secara teori komputer ini memiliki satu *processor* dan mengeksekusi beberapa instruksi secara paralel.

1.3.4 MIMD

Komputer berarsitektur ini paling banyak digunakan untuk membangun *super computer*. Komputer ini memiliki lebih dari satu *processors* dan mengeksekusi lebih dari satu instruksi secara paralel.

1.4 *SERVER* YANG DIGUNAKAN

Pada praktikum kali ini, saya tidak bisa menggunakan *server HPC* yang disediakan oleh ITB karena masalah koneksi. Oleh karena itu, saya menggunakan *server* lain agar bisa menduplikasi apa yang seharusnya dikerjakan di *server* ITB.

Saya menggunakan *server virtual machine* milik *Google Cloud*².

Berikut adalah spesifikasinya:

Server ini bisa diakses menggunakan *command line* menggunakan **ssh** langsung ke *IP Public* yang diberikan *Google*.

Server ini berjalan di *operating system* Ubuntu Linux 20.04 LTS.

Berikut adalah tampilan hasil **lscpu**:

Berikut adalah tampilan hasil **htop**:

2 *METHOD*

Pada praktikum ini, kita akan melakukan *parallel processing* menggunakan **Python** versi 3.8.10 di komputer berbasis **Linux Ubuntu OS**. Ada beberapa metode *parallel processing* yang hendak dilakukan, yakni:

1. *Broadcast*,
 1. *Broadcast-gather*,
 2. *Broadcast-reduce*,

²<https://ikanx101.com/blog/vm-cloud/>

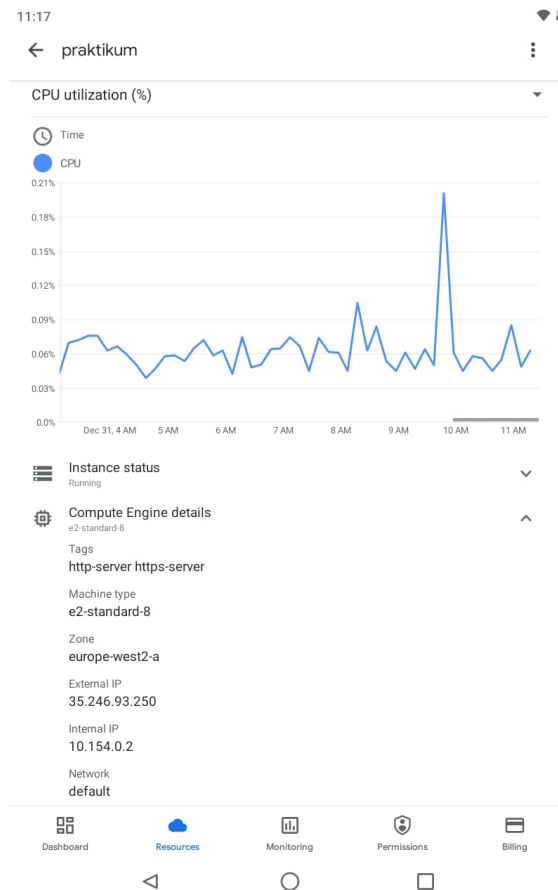


Figure 4: Spesifikasi Server yang Digunakan

```

lx@localhost:~/209_1TB/Semester 1/Jaringan dan Pengolahan Data Paralel/praktikum$ ssh 35.246.93.250
Enter passphrase for key '/home/lx/.ssh/id_rsa':
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1023-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Fri Dec 31 03:38:00 UTC 2021

System load:  0.16          Processes:      158
Usage of /:   48.4% of 9.52GB Users logged in:   0
Memory usage: 1%           IPv4 address for ens4: 10.154.0.2
Swap usage:   0%

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.
   https://ubuntu.com/blog/microk8s-memory-optimisation

0 updates can be applied immediately.

Last login: Fri Dec 31 02:40:18 2021 from 111.94.196.248
lx@praktikum:~$

```

Figure 5: Tampilan Awal Setelah Login ssh

```

Last login: Fri Dec 31 02:40:18 2021 from 111.94.196.248
l@praktikum:~$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:             Little Endian
Address sizes:          46 bits physical, 48 bits virtual
CPU(s):                 8
On-line CPU(s) list:   0-7
Thread(s) per core:    2
Core(s) per socket:    4
Socket(s):              1
NUMA node(s):           1
Vendor ID:              GenuineIntel
CPU family:             6
Model:                  79
Model name:             Intel(R) Xeon(R) CPU @ 2.20GHz
Stepping:               0
CPU MHz:                2200.202
BogoMIPS:               4400.40
Hypervisor vendor:      KVM
Virtualization type:     full
L1d cache:              128 KiB
L1i cache:              128 KiB
L2 cache:               1 MiB
L3 cache:               55 MiB
NUMA node0 CPU(s):      0-7
Vulnerability Itlb multihit: Not affected
Vulnerability L1tf:      Mitigation: PTE Inversion
Vulnerability Mds:       Mitigation: Clear CPU buffers; SMT Host state unknown
Vulnerability Meltdown:  Mitigation: PTI
Vulnerability Spec store bypass: Mitigation: Speculative Store Bypass disabled via prctl and seccomp
Vulnerability Spectre v1: Mitigation: usercopy/swapgs barriers and __user pointer sanitization
Vulnerability Spectre v2: Mitigation: Full generic retpoline, IBPB conditional, IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds:     Not affected
Vulnerability Tsx async abort: Mitigation: Clear CPU buffers; SMT Host state unknown

```

Figure 6: lscpu dari Server

```

1 [ 0.7%] 5 [ 0.0%]
2 [ 0.0%] 6 [ 0.0%]
3 [ 0.0%] 7 [ 0.0%]
4 [ 0.0%] 8 [ 0.0%]
Mem [|||||] 264M/31.4G Tasks: 29, 64 thr; 1 running
Swap [ ] 0K/0K Load average: 0.00 0.00 0.00
Uptime: 1 day, 20:21:08

  PID USER      PRI  NI  VIRT   RES   SHR  S CPU% MEM%   TIME+  Command
 61686 ix      20    0 8292 3576 3120  R  2.6  0.0  0:00.11 htop
 215 root      19   -1 76080 15948 14728  S  0.0  0.0  0:03.04 /lib/systemd/systemd-journald
 247 root      20    0 18944 5488 4080  S  0.0  0.0  0:00.56 /lib/systemd/systemd-udev
 378 root      RT   0 273M 17996 8204  S  0.0  0.1  0:01.34 /sbin/multipathd -d -s
 379 root      RT   0 273M 17996 8204  S  0.0  0.1  0:00.00 /sbin/multipathd -d -s
 380 root      RT   0 273M 17996 8204  S  0.0  0.1  0:00.21 /sbin/multipathd -d -s
 381 root      RT   0 273M 17996 8204  S  0.0  0.1  0:08.86 /sbin/multipathd -d -s
 382 root      RT   0 273M 17996 8204  S  0.0  0.1  0:00.00 /sbin/multipathd -d -s
 383 root      RT   0 273M 17996 8204  S  0.0  0.1  0:00.00 /sbin/multipathd -d -s
 377 root      RT   0 273M 17996 8204  S  0.0  0.1  0:19.48 /sbin/multipathd -d -s
 482 systemd-n 20    0 26612 7612 6756  S  0.0  0.0  0:01.15 /lib/systemd/systemd-networkd
 485 systemd-r 20    0 23904 12060 8136  S  0.0  0.0  0:01.29 /lib/systemd/systemd-resolved
 609 root      20    0 235M 9376 8344  S  0.0  0.0  0:03.62 /usr/lib/accounts-service/accou
 824 root      20    0 235M 9376 8344  S  0.0  0.0  0:00.04 /usr/lib/accounts-service/accou
 608 root      20    0 235M 9376 8344  S  0.0  0.0  0:03.76 /usr/lib/accounts-service/accou
 627 messagebu 20    0 4052 4860 3896  S  0.0  0.0  0:01.54 /usr/bin/dbus-daemon -system
 634 root      20    0 1274M 22900 16556 S  0.0  0.1  0:03.98 /usr/bin/google_osconfig_agent
 635 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.96 /usr/bin/google_osconfig_agent
 636 root      20    0 1274M 22900 16556 S  0.0  0.1  0:00.00 /usr/bin/google_osconfig_agent
 637 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.62 /usr/bin/google_osconfig_agent
 639 root      20    0 1274M 22900 16556 S  0.0  0.1  0:00.00 /usr/bin/google_osconfig_agent
 642 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.51 /usr/bin/google_osconfig_agent
 644 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.46 /usr/bin/google_osconfig_agent
 645 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.73 /usr/bin/google_osconfig_agent
 646 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.50 /usr/bin/google_osconfig_agent
 647 root      20    0 1274M 22900 16556 S  0.0  0.1  0:01.68 /usr/bin/google_osconfig_agent

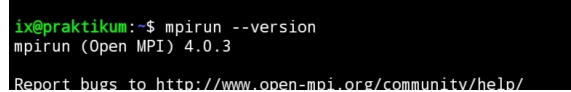
```

Figure 7: htop dari Server

2. *Scatter*,
 1. *Scatter-reduce*,
 2. *Scatter-gather*,
3. *Gather*,
4. *Reduce*,
5. *Multi-processing*,
6. *Multi-thread*,
7. *Point-to-point*.

Kemudian semua metode *parallel processing* ini akan dibandingkan *runtime*-nya dengan *serial processing*.

2.1 MPI



```
ix@praktikum:~$ mpirun --version
mpirun (Open MPI) 4.0.3
Report bugs to http://www.open-mpi.org/community/help/
```

Figure 8: Versi MPI yang Digunakan

penjelasan tentang MPI, server google, midpoint, lalu metode monte carlo yang digunakan.

3 *RESULT AND DISCUSSION*

3.1 Soal I

3.2 Soal II

Perhitungan π menggunakan rumus: $4 \times \text{sqrt}1 - x^2$

3.3 Soal III

4 *CONCLUSION*

lalala (Hillier and Lieberman 2001)

REFERENCES

Hillier, Frederick S., and Gerald J. Lieberman. 2001. *Introduction to Operations Research*. 7th ed. New York, US: McGraw Hill. www.mhhe.com.