

KUMPULAN TUGAS KOMPUTASI LANJUT DAN BIG DATA

untuk memenuhi Ujian Akhir Semester
Mata Kuliah Komputasi Lanjut dan Big Data

Dosen Pengampu:
Prof Alhadi Bustaman, Ph.D.



disusun oleh:
Cintya Kusuma Mahadhika 2206014725
Elizabeth Lilies Megawati 2206014731
Eka Dwi Agustina Ginting 2206103390

Program Magister Matematika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Indonesia
2022

Daftar Isi

Halaman Judul	1
Daftar Isi	2
Kata Pengantar	5
BAB 1	
Perbandingan Komputasi Serial dan Komputasi Paralel	6
BAB 2	
Komputasi Paralel: Taksonomi Flynn dan Sistem Memori	20
BAB 3	
Studi Kasus Pemrograman Paralel dan Taksonomi Flynn	40
BAB 4	
Aplikasi Parallel Computing (Shared Memory, Distributed Memory, dan Hybrid Memory)	69
BAB 5	
Analisis Algoritma Paralel	108
BAB 6	
Implementasi Algoritma Paralel di Beberapa Platform Bahasa Pemrograman	127
Tugas UTS Elizabeth Lilies Megawati 2206014731	143
Tugas UTS Cintya Kusuma Mahadhika 2206014725	155
Tugas UTS Eka Dwi Agustina Ginting 2206103390	168

Kata Pengantar

Segala puji bagi Allah yang telah memberikan kemudahan dan kesempatan kepada kami untuk menyusun buku ini. Kami mengucapkan terima kasih kepada-Nya atas bimbingan dan pertolongan-Nya selama proses penyusunan buku ini.

Sebagai pengantar untuk buku kumpulan tugas komputasi lanjut dan big data ini, kami ingin menyampaikan terima kasih kepada semua pihak yang telah membantu dalam proses penyusunan buku ini.

Pertama, dengan segala hormat, kami mengucapkan terima kasih yang sebesar-besarnya kepada Dosen yang telah memberikan bimbingan dan arahan yang sangat bermanfaat dalam mata kuliah komputasi lanjut dan big data ini. Kami sangat menghargai dedikasi dan komitmen Dosen dalam menyampaikan materi yang berkualitas serta membantu kami dalam memahami materi pembelajaran komputasi lanjut dan big data yang benar dan baik.

Kami juga sangat berterima kasih atas waktu yang Dosen khususkan untuk memberikan arahan dan bimbingan kepada kami secara individual. Kami yakin bahwa dengan bimbingan dan arahan yang telah diberikan oleh Dosen, kami akan mampu menyusun makalah tugas komputasi lanjut dan big data yang berkualitas dan dapat memberikan manfaat bagi masyarakat. Terima kasih atas perhatian dan bantuan yang telah diberikan oleh Dosen. Kami berharap dapat terus memperoleh dukungan dan bimbingan dari Dosen di masa yang akan datang.

Kedua, kami mengucapkan terima kasih kepada para penulis yang telah memberikan sumbangsihnya dalam bentuk makalah yang terkumpul dalam buku ini. Kami menghargai dedikasi dan komitmen mereka dalam menyusun makalah yang berkualitas.

Dengan adanya buku ini, kami harap dapat memberikan manfaat bagi para penulis dan pembaca yang akan mempelajarinya. Kami berharap buku ini dapat memberikan wawasan baru bagi pembaca dan menjadi sumber inspirasi bagi para penulis untuk terus berkarya dan mengembangkan ilmu pengetahuan di bidangnya masing-masing. Terima kasih atas perhatian dan waktu yang Anda berikan untuk membaca buku ini. Kami berharap dapat memberikan pengalaman membaca yang menyenangkan bagi Anda.

Salam,
Tim Penyusun Buku

BAB 1

Perbandingan Komputasi Serial dan Komputasi Paralel

Cintya Kusuma Mahadhika, Eka Dwi Agustina Ginting, Elizabeth Lilies Megawati

Abstrak

Ada beberapa masalah dalam komputasi yang dapat diselesaikan dengan komputasi serial, namun beberapa masalah seperti pengelolaan *big data* membutuhkan teknik komputasi yang lebih efektif dan efisien. Kurangnya pengetahuan tentang perbedaan komputasi ini dapat menghambat proses penyelesaian masalah komputasi. Oleh karena itu, dalam *paper* ini dijabarkan definisi, kelebihan, maupun kekurangan antara komputasi serial dan komputasi paralel. Dari pemaparan di *paper* ini, dapat disimpulkan bahwa komputasi serial atau sekuensial akan menyelesaikan masalah dengan melakukan setiap langkah secara berurutan dan mendapatkan sebuah hasil apabila langkah sebelumnya telah dilakukan. *Hardware* komputasi serial relatif lebih murah, namun penyelesaian masalah dalam komputasi serial relatif lebih lama dibanding komputasi paralel. Komputasi paralel sendiri adalah proses komputasi yang membagi beban instruksi ke dalam beberapa bagian kecil sub proses instruksi, kemudian sub bab instruksi tersebut dijalankan pada prosesor yang berbeda secara bersamaan dalam menyelesaikan masalah komputasi. Waktu eksekusi dalam komputasi paralel memang relatif lebih cepat, namun *hardware* untuk komputasi paralel cenderung lebih mahal.

Kata kunci: komputasi, komputasi paralel, komputasi serial

PENDAHULUAN

Saat komputer pertama kali dikembangkan, komputasi yang dikenal adalah komputasi serial. Semakin banyak program yang harus dieksekusi, makin lama pula waktu yang dibutuhkan oleh komputasi serial (Sternberg, 1966). Oleh karena itu, dikembangkan metode komputasi paralel agar proses komputasi menjadi lebih efisien. Akan tetapi, menurut New Mexico State University (2021), tidak semua masalah cocok diselesaikan dengan komputasi paralel. Pada kasus seperti perhitungan sederhana, penggunaan komputasi paralel justru tidak efisien. Oleh karena itu, perlu adanya pengetahuan mengenai perbedaan komputasi serial dan paralel.

Dalam *paper* ini, akan dipaparkan perbedaan mengenai komputasi serial dan paralel serta kelebihan dan kekurangan masing-masing proses. Dengan demikian, diharapkan pembaca

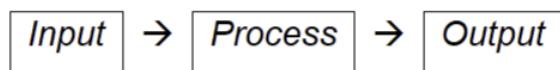
dapat memilih metode komputasi yang tepat untuk menyelesaikan masalah yang dimiliki.

LANDASAN TEORI

1. Komputasi

Dalam mencapai target pekerjaan yang akan dicapai, kita harus memahami permasalahan dan alur pekerjaan yang jelas. Hal tersebut juga berlaku pada pemrograman yang tidak terlepas dari konsep kerja sebuah komputer atau kita dapat mengenalnya dengan sebutan komputasi. Komputasi merupakan suatu proses pemecahan masalah dari sebuah data masukan (input) kemudian diperoleh data keluaran (output) dengan menggunakan algoritma yang telah ditentukan. (dalam Rahmatullah, dkk, 2021)

Algoritma menurut Sitorus (2015) adalah susunan langkah penyelesaian suatu masalah secara sistematis dan logis.



Gambar 1 Prinsip Kerja Algoritma

Sitorus (2015) juga mengungkapkan salah satu tahapan pokok untuk menghasilkan program untuk menyelesaikan masalah adalah memahami permasalahan dan tujuan sebuah program dibuat. Hal ini berarti kita harus mampu mengidentifikasi jenis, bentuk, dan karakteristik input serta output yang diharapkan. Pada permasalahan yang besar, kita juga perlu mengetahui asal, frekuensi dan volume data input serta tujuan. Dengan begitu, kita dapat menentukan proses komputasi yang dapat digunakan untuk menyelesaikan permasalahan yang sudah diidentifikasi. Proses komputasi yang dimaksud adalah komputasi serial dan komputasi paralel.

2. Komputasi Serial

Komputasi serial pertama digunakan dalam mesin Turing (Dawson, 2020). Komputasi ini memproses satu operasi dalam satu waktu. Dari sini, kita bisa mengetahui bahwa komputasi serial muncul lebih dulu dibanding komputasi paralel.

Komputasi serial atau komputasi sekuensial merupakan sebuah proses untuk melakukan pemecahan masalah dengan melakukan setiap langkah secara berurutan dan mendapatkan sebuah hasil apabila langkah sebelumnya telah dilakukan (Gebali, 2011).

Jika urutan penulisan diubah, maka kemungkinan akan memberikan hasil akhir yang berbeda. Komputasi sekuensial menggunakan memori internal atau perangkat antarmuka sebagai media untuk menerima masukan dan mengirim keluaran. Contoh prosesor yang masih menggunakan metode komputasi ini adalah Intel Pentium 3 dan Pentium 4.

Jika komputasi serial digunakan untuk mengeksekusi program yang besar, maka akan muncul beberapa masalah. Karena komputasi serial akan memecah tugas tersebut menjadi beberapa langkah lalu menjalankannya satu per satu, maka menurut Rastogi dan Zaheer, kemungkinan yang terjadi adalah:

- 1) Membutuhkan waktu yang lebih lama.

Semakin besar tugas yang dieksekusi, tentu semakin banyak program yang harus dijalankan. Karena komputasi serial mengeksekusi program tersebut satu per satu, maka akan semakin lama waktu yang dibutuhkan.

- 2) Kinerja prosesor akan semakin berat.

Karena hanya ada satu prosesor yang mengeksekusi program, maka tentu beban prosesor akan semakin berat jika harus mengeksekusi program yang besar.

Karena itu, komputasi serial cocok digunakan untuk mengeksekusi program yang sederhana seperti menyelesaikan perhitungan sederhana atau menyelesaikan masalah komunikasi di MODEM (Sweet, 1999). Dalam masalah tersebut, komputer akan mengambil sinyal, menerjemahkan menjadi *number of bytes*, kemudian dipilih untuk ditransmisikan. Contoh mesin yang melakukan pemrosesan serial, yaitu mesin pentium 3 dan pentium 4. (Lithmee, 2019)

Seiring berjalananya waktu, manusia memerlukan penyelesaian untuk masalah yang semakin kompleks seperti penjadwalan dalam jumlah yang besar maupun membangun pesawat. Dengan alur komputasi serial seperti yang dipaparkan, tentu membutuhkan waktu lama untuk mengeksekusi masalah-masalah tersebut. Oleh karena itu, muncullah komputasi paralel.

3. Komputasi Paralel

Komputasi paralel (Gebali, 2011) merupakan pengembangan dari komputasi serial dimana komputasi ini digunakan ketika kapasitas yang diperlukan sangat besar, baik karena harus mengelola data dalam jumlah yang sangat besar atau karena proses komputasi yang banyak. Proses komputasi ini adalah membagi beban instruksi ke dalam beberapa bagian kecil sub proses instruksi, kemudian sub bab instruksi tersebut

dijalankan pada prosesor yang berbeda secara bersamaan dalam menyelesaikan masalah komputasi. Perlu ditekankan bahwa komputasi paralel berbeda dengan *multitasking* yang mana prosesor tunggal mengeksekusi beberapa tugas secara bersamaan. Dengan demikian, program akan dieksekusi dengan menggunakan *multiprocessor*. Contoh prosesor yang mendukung komputasi ini adalah Intel Core i5 dan i7.

Komputasi paralel bisa menyelesaikan masalah dengan lebih efisien dibanding komputasi serial, namun bukan berarti komputasi paralel tidak memiliki kekurangan. Dibanding komputasi serial, komputasi paralel menggunakan daya yang lebih besar karena harus mengeksekusi beberapa program bersamaan. Selain itu, rata-rata *hardware* untuk komputasi paralel juga lebih mahal. Kemudian, karena komputasi paralel akan memecah masalah menjadi beberapa program untuk dieksekusi bersama, maka komputasi paralel tidak akan menyelesaikan masalah yang membutuhkan hasil dari iterasi sebelumnya seperti perhitungan deret Fibonacci.

Komputasi paralel menggunakan shared memory atau *processor* lain melalui jaringan interkoneksi antar *processor* atau jaringan komputer sebagai media untuk menerima masukan dan mengembalikan keluaran. Sumber daya komputasi dapat berupa sebuah komputer tunggal dengan beberapa prosesor, beberapa komputer yang terhubung dalam jaringan (distributed computing), maupun kombinasi dari keduanya.

Guntara (2016) menyebutkan bahwa Michael J. Flynn membagi komputer dalam 4 kategori yaitu :

a. ***Single Instruction, Single Data,***

Satu-satunya komputer yang menggunakan arsitektur Von Neumann karena pada model ini hanya digunakan 1 processor saja dengan setiap instruksi dieksekusi secara berurut. Model ini bisa dikatakan sebagai model untuk komputasi tunggal karena keseluruhan mesin SISD memanfaatkan program *counter* yang melakukan sederetan eksekusi instruksi secara serial. Setiap masing-masing instruksi diambil dari memori, program counter akan selalu diupdate untuk mengisi alamat instruksi yang akan diambil dan dieksekusikan secara berurutan. Beberapa contoh komputer yang menggunakan model SISD adalah UNIVAC1, IBM 360, CDC 7600, Cray 1 dan PDP.

b. ***Single Instruction, Multiple Data,***

SIMD menggunakan banyak processor dengan instruksi yang sama, namun setiap processor mengolah data yang berbeda. Sebagai contoh kita ingin mencari angka 27 pada deretan angka yang terdiri dari 100 angka, dan kita menggunakan 5

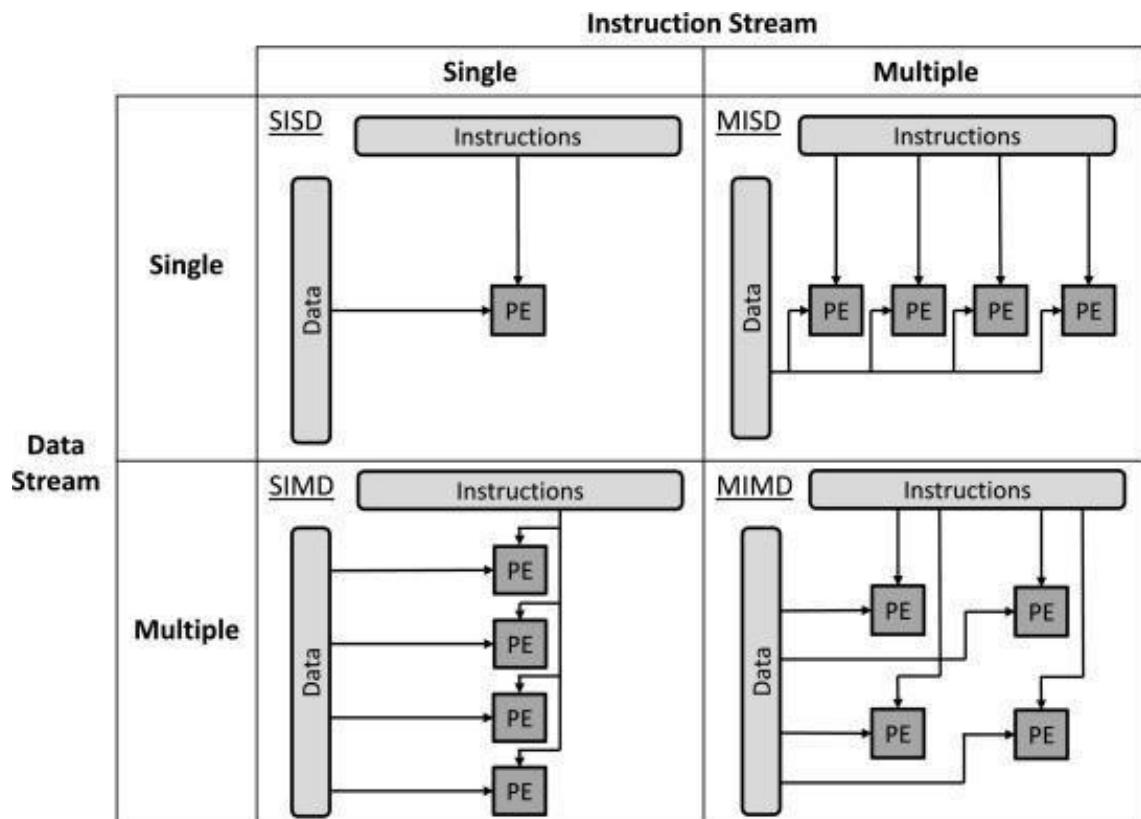
processor. Pada setiap processor kita menggunakan algoritma atau perintah yang sama, namun data yang diproses berbeda. Misalnya processor 1 mengolah data dari deretan / urutan pertama hingga urutan ke 20, processor 2 mengolah data dari urutan 21 sampai urutan 40, begitu pun untuk processor-processor yang lain. Beberapa contoh komputer yang menggunakan model SIMD adalah ILLIAC IV, MasPar, Cray X-MP, Cray Y-MP, Thinking Machine CM-2 dan Cell Processor (GPU).

c. ***Multiple Instruction, Single Data.***

MISD menggunakan banyak processor dengan setiap processor menggunakan instruksi yang berbeda namun mengolah data yang sama. Hal ini merupakan kebalikan dari model SIMD. Untuk contoh, kita bisa menggunakan kasus yang sama pada contoh model SIMD namun cara penyelesaian yang berbeda. Pada MISD jika pada komputer pertama, kedua, ketiga, keempat dan kelima sama-sama mengolah data dari urutan 1-100, namun algoritma yang digunakan untuk teknik pencarinya berbeda di setiap processor. Sampai saat ini belum ada komputer yang menggunakan model MISD.

d. ***Multiple Instruction, Multiple Data.***

MIMD menggunakan banyak processor dengan setiap processor memiliki instruksi yang berbeda dan mengolah data yang berbeda. Namun banyak komputer yang menggunakan model MIMD juga memasukkan komponen untuk model SIMD. Beberapa komputer yang menggunakan model MIMD adalah IBM POWER5, HP/Compaq AlphaServer, Intel IA32, AMD Opteron, Cray XT3 dan IBM BG/L.



Gambar 2. Ilustrasi Sistem Taksonomi Flynn

Sumber <https://www.sciencedirect.com/topics/computer-science/>

Komputer sekuensial berdasarkan klasifikasi Flynn adalah kelompok komputer SISD hanya mempunyai satu unit pengendali untuk menentukan instruksi yang akan dieksekusi. Pada setiap satuan waktu hanya satu instruksi yang dapat dieksekusi, dimana kecepatan akses ke memori dan kecepatan piranti masukan dan keluaran dapat memperlambat proses komputasi.

Beberapa metoda dibangun untuk menghindari masalah tersebut, seperti penggunaan cache memory. Namun komputer sekuensial ini tetap mengalami keterbatasan jika menangani masalah yang memerlukan kecepatan tinggi. Hal-hal tersebut di atas pada akhirnya melatarbelakangi lahirnya sistem komputer paralel. Berdasarkan klasifikasi Flynn, komputer paralel termasuk kelompok SIMD atau MIMD.

4. Contoh pemrograman

Berikut ini akan diberikan contoh pemrograman dengan menggunakan komputasi paralel dan serial dengan Python dalam penjumlahan 1000000 bilangan asli pertama.

a. Dengan komputasi paralel

1) Kode

```
import time

# get the start time
st = time.time()

from multiprocessing import Pool

def sum_nums(args):
    low = int(args[0])
    high = int(args[1])
    return sum(range(low,high+1))

if __name__ == "__main__":
    n = 1000000
    procs = 2
    sizeSegment = n/procs

    # Create size segments list
    jobs = []
    for i in range(0, procs):
        jobs.append((i*sizeSegment+1, (i+1)*sizeSegment))
    pool = Pool(procs).map(sum_nums, jobs)
    result = sum(pool)
    print(result)

    # get the end time
    et = time.time()
    # get the execution time
    elapsed_time = et - st
    print('Execution time:', elapsed_time, 'seconds')
```

b) Hasil dan waktu berjalannya program

Dari kode tersebut, diperoleh hasil 500000500000 dengan waktu eksekusi 0.051290273666381836 detik

b. Dengan komputasi serial

1) Kode

```
import time  
  
# get the start time  
st = time.time()  
num = 1000000  
  
if num < 0:  
    print("Enter a positive number")  
else:  
    sum = 0  
  
    # use while loop to iterate until zero  
    while(num > 0):  
        sum += num  
        num -= 1  
  
    print(sum)  
  
    # get the end time  
    et = time.time()  
  
    # get the execution time  
    elapsed_time = et - st  
  
    print('Execution time:', elapsed_time, 'seconds')
```

2) Hasil dan waktu berjalannya program

Dari kode tersebut, diperoleh hasil 5000050000 dengan waktu eksekusi 0.20639348030090332 detik

Dari contoh tersebut, terlihat bahwa komputasi paralel dapat menyelesaikan masalah perhitungan jumlah 1000000 bilangan asli dengan lebih cepat. Namun, tidak semua masalah komputasi selalu lebih cepat diselesaikan dengan komputasi paralel. Sebagai contoh, akan diberikan kode Python untuk menyelesaikan masalah perhitungan 16 bilangan asli pertama.

c. Dengan komputasi paralel

1) Kode

```
import time
# get the start time
st = time.time()
from multiprocessing import Pool
def sum_nums(args):
    low = int(args[0])
    high = int(args[1])
    return sum(range(low,high+1))
if __name__ == "__main__":
    n = 16
    procs = 2
    sizeSegment = n/procs
    # Create size segments list
    jobs = []
    for i in range(0, procs):
        jobs.append((i*sizeSegment+1,(i+1)*sizeSegment))
    pool = Pool(procs).map(sum_nums, jobs)
    result = sum(pool)
    print(result)
    # get the end time
    et = time.time()
    # get the execution time
    elapsed_time = et - st
    print('Execution time:', elapsed_time, 'seconds')
```

2) Hasil dan waktu berjalannya program

Dari kode tersebut, diperoleh hasil 136 dengan waktu eksekusi 0.038732290267944336 detik

d. Dengan komputasi serial

1) Kode

```
import time  
  
# get the start time  
st = time.time()  
  
num = 16  
  
if num < 0:  
    print("Enter a positive number")  
else:  
    sum = 0  
  
    # use while loop to iterate until zero  
    while(num > 0):  
        sum += num  
        num -= 1  
  
        print(sum)  
  
    # get the end time  
    et = time.time()  
  
    # get the execution time  
    elapsed_time = et - st  
  
    print('Execution time:', elapsed_time, 'seconds')
```

2) Hasil dan waktu berjalannya program

Dari kode tersebut, diperoleh hasil 136 dengan waktu eksekusi 0.000060558319091796875 detik

Dari contoh yang diberikan, terlihat bahwa komputasi serial justru menyelesaikan masalah jumlah 16 bilangan asli pertama lebih cepat dibanding komputasi paralel. Dengan demikian, dapat disimpulkan bahwa tidak semua masalah komputasi bisa diselesaikan oleh komputasi paralel dengan efisien.

PEMBAHASAN

Komputasi merupakan proses untuk menjalankan suatu algoritma. Terdapat dua komputasi, yaitu:

- a. komputasi serial, komputasi yang digunakan untuk mengeksekusi program sederhana dengan cara instruksi-instruksi dikerjakan secara berurutan untuk mendapatkan suatu hasil.
- b. Komputasi paralel, komputasi yang digunakan untuk mengeksekusi program besar dengan cara memecah seluruh instruksi menjadi bagian yang terkecil kemudian kemudian setiap bagian dieksekusi dengan dua atau lebih prosesor/memori secara bersamaan.

Komputasi paralel menjadi salah satu solusi komputasi dalam skala besar yang menawarkan kecepatan pemrosesan dengan cara. Prosesor-prosesor yang bekerja pada komputasi paralel akan mengoptimalkan *resource* pada sistem komputer untuk mencapai tujuan yang sama, sehingga waktu yang dibutuhkan relatif lebih singkat dibandingkan pemrosesan dengan komputasi serial. Beberapa alasan lainnya yang menjadikan suatu program lebih efektif dan efisien jika menggunakan komputasi paralel, antara lain: sumber daya komputasi yang ada belum cukup mampu untuk mendukung penyelesaian menggunakan komputasi serial; adanya sumber daya non-lokal yang dapat digunakan melalui jaringan atau internet; penghematan biaya pengadaan perangkat keras; dan adanya keterbatasan kapasitas memori pada mesin untuk komputasi serial.

KESIMPULAN

Dari landasan teori dan pembahasan, dapat disimpulkan perbedaan antara komputasi paralel dan komputasi serial dalam tabel berikut.

Tabel 1 Perbedaan Komputasi Serial dan Komputasi Paralel

Pembeda	Komputasi Serial	komputasi Paralel
Jenis pengolahan	satu instruksi selesai pada satu waktu dan semua instruksi dieksekusi oleh sebuah prosesor secara berurutan	beberapa tugas dapat selesai pada waktu bersamaan oleh prosesor-prosesor yang berbeda
Jumlah prosesor	satu prosesor	banyak prosesor
Beban kerja	lebih berat	lebih ringan
Waktu yang dibutuhkan	lebih lama	lebih singkat
Harga	lebih murah	lebih mahal

Tautan *Youtube*: https://youtu.be/_bBHttfQpoQ

DAFTAR PUSTAKA

- Dawson, Michael R.W. 2020. *Serial versus Parallel Processing*. Athabasca: Athabasca University Press.
- Gebali, Fayez. 2011. *Algorithms and Parallel Computing*. John Wiley & Sons, Inc., Hoboken, New Jersey Published simultaneously in Canada.
- Guntara, Achmad. 2016. *Hubungan Pararel Processing dengan Komputasi Modern*. diakses dari <https://guntaraachmad.wordpress.com/2016/04/29/hubungan-pararel-processing-dengan-komputasi-modern/> pada 21 september 2022
- Rahmatullah, Hidayat, dan Zukarnain. 2021. *Analisis Perbandingan Performa Pemrograman Sekuensial Dan Paralel Dengan Skema Uji Matrix, Filter Dan Quick Sort*. Jurnal Teknologi Informasi Universitas Lambung Mangkurat. diakses pada 13 September 2022 dari https://www.researchgate.net/publication/351300511_Analisis_Perbandingan_Performa_Pemrograman_Sekuensial_Dan_Paralel_Dengan_Skema_Uji_Matrix_Filter_Dan_Quick_Sort
- <http://journal.uii.ac.id/index.php/Snati/article/download/1607/1382>
- <https://www.sciencedirect.com/topics/computer-science/>
- Lithmee, 2019. *What is the Difference Between Serial and Parallel Processing in Computer Architecture*. diakses dari <https://pediaa.com/what-is-the-difference-between-serial-and-parallel-processing-in-computer-architecture/> diakses tanggal 27 September 2022
- Rastogi, Shubhangi & Zaheer, Hira. 2018. *Quality, IT and Business Operations*. Singapore: Springer Singapore.
- Serial vs Parallel Jobs* (14 September 2022). Diakses di <https://hpc.nmsu.edu/discovery/slurm/serial-parallel-jobs/>
- Sitorus, Lamhot. 2015. *ALGORITMA DAN PEMROGRAMAN*. Yogyakarta: ANDI. diakses pada 13 September 2022 dari https://www.google.co.id/books/edition/Algoritma_dan_

[Pemrograman/MRHwCgAAQBAJ?hl=en&gbpv=1&dq=algoritma+dan+pemrograman+simulator&pg=PR6&printsec=frontcover](#)

Sternberg, S. (1966). *High Speed Scanning in Human Memory*. Science, 153, 652–654.

Sweet, Michael. 1999. *Serial Programming Guide for POSIX Operating Systems*.
<https://www.cmrr.umn.edu/~strupp/serial.html#CONTENTS>

Kartawijaya, Suparman. 2021. *Parallel Processing. (Sistem Terdistribusi)*. diakses pada 13 September 2022. dari <https://docplayer.info/197377439-Parallel-processing-sistem-terdistribusi.html>

BAB 2

Komputasi Paralel: Taksonomi Flynn dan Sistem Memori

Cintya Kusuma Mahadhika, Elizabeth Lilies Megawati, Eka Dwi Agustina Ginting

ABSTRAK

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer secara bersamaan. Dalam komputasi paralel, dikenal beberapa sistem penyimpanan seperti *shared memory*, *distributed memory*, dan *hybrid*. Masing-masing sistem penyimpanan memiliki karakternya sendiri sehingga pengguna dapat memilih sistem sesuai kebutuhannya.

Kata kunci: Komputasi paralel, taksonomi Flynn, *shared memory*, *distributed memory*, *hybrid memory*.

PENDAHULUAN

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer (Gebali, 2011). Dalam komputasi, termasuk komputasi paralel, dikenal memori sebagai perangkat atau sistem yang digunakan untuk menyimpan informasi untuk digunakan segera di komputer atau perangkat keras komputer terkait dan perangkat elektronik digital (Hemmendinger, 2016). Sistem memori dalam komputasi ada berbagai macam dan memiliki karakter masing-masing begitupun dengan kelebihan dan kekurangannya (Ali & Syed, 2011).

Oleh karena itu, dalam makalah ini akan dipaparkan lebih lanjut mengenai komputasi paralel dan sistem memori yang digunakan. Dengan demikian, pembaca diharapkan dapat menentukan sistem memori yang tepat sesuai dengan kebutuhannya.

PEMBAHASAN

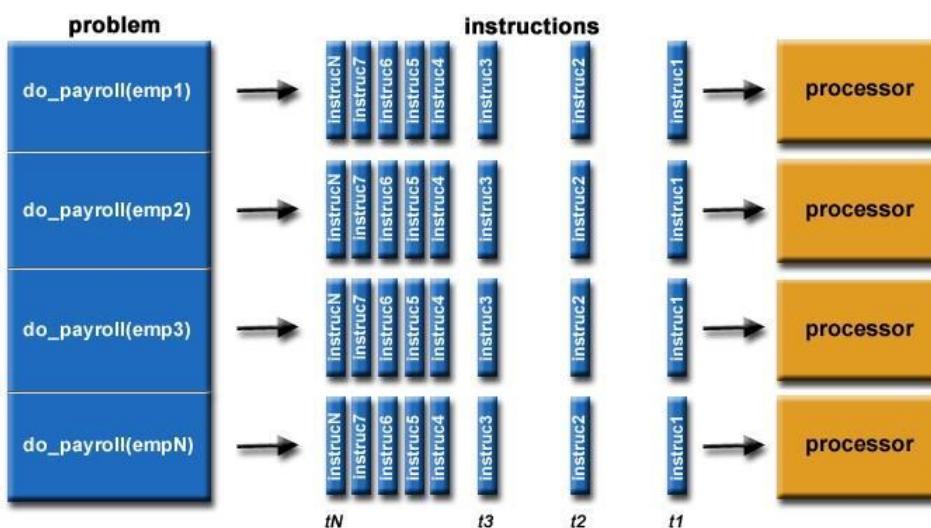
1. Komputasi Paralel

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer(Gebali, 2011). Biasanya diperlukan saat kapasitas yang diperlukan sangat besar, baik karena harus mengolah data dalam jumlah besar ataupun karena tuntutan proses komputasi yang banyak. Untuk melakukan aneka jenis komputasi paralel ini diperlukan infrastruktur mesin paralel yang terdiri dari banyak komputer yang dihubungkan dengan jaringan dan mampu bekerja secara paralel untuk menyelesaikan satu masalah. Untuk itu diperlukan aneka perangkat lunak pendukung yang biasa disebut sebagai middleware yang berperan untuk mengatur

distribusi pekerjaan antar node dalam satu mesin paralel. Selanjutnya pemakai harus membuat pemrograman paralel untuk merealisasikan komputasi.

Pemecahan suatu masalah komputasi dapat dilakukan dengan (Atiq, (n.d)):

1. Membagi masalah menjadi beberapa bagian diskrit yang dapat diselesaikan secara bersamaan.
2. Setiap bagian selanjutnya dipecahkan menjadi serangkaian instruksi.
3. Instruksi dari setiap bagian mengeksekusi secara bersamaan pada processor yang berbeda.
4. Terjadi mekanisme kontrol/koordinasi.



Gambar 1. Ilustrasi pemecahan masalah komputasi

Sumber: <https://pdfcoffee.com/tested-pdf-free.html>

Dalam Gebali (2011) paralelisme dalam suatu komputer dapat diaplikasikan pada beberapa tingkatan, seperti berikut.

- a. *Data-level parallelism*, dimana secara bersama-sama beroperasi pada banyak bit datum atau padbanyak data. Contohnya adalah penambahan bit paralel, perkalian, dan pembagian bilangan biner, pengolah vektor, dan array sistolik untuk menangani beberapa smpel data.
- b. *Instruction-level parallelism*, yaitu secara bersama-sama mengeksekusi lebih dari satu instruksi dalam prosesor. Contohnya penggunaan instruksi *pipelining*.
- c. *Thread-level parallelism*. *Thread* adalah bagian dari program yang berbagi sumber daya prosesor dengan *thread* yang lain. Pada *TLP*, beberapa *thread* perangkat lunak dieksekusi secara simultan pada satu atau beberapa prosesor.

d. *Process-level parallelism.* Sebuah proses adalah program yang berjalan dalam komputer. Suatu proses mencadangkan sumber daya komputernya sendiri. Ini tentu saja multitasking klasik dan komputasi *time-sharing* dimana beberapa program berjalan secara simultan pada satu atau beberapa mesin.

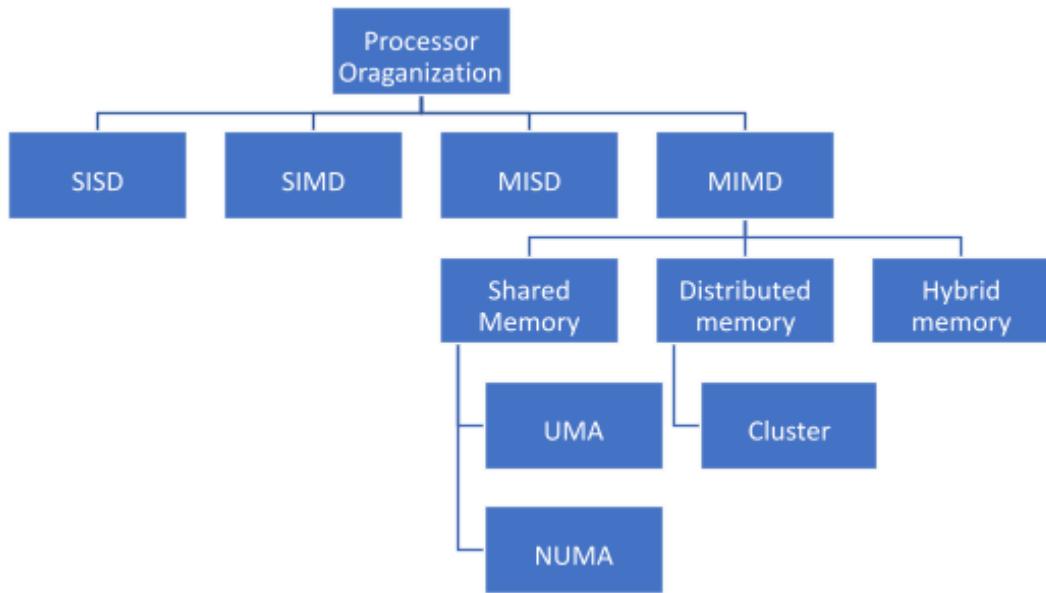
2. Taksonomi Flynn

Arsitektur Komputer Paralel adalah metode pengorganisasian semua sumber daya untuk memaksimalkan kinerja dan kemampuan program dalam batas-batas teknologi dan biaya. (tutorialspoint.com). Sekumpulan elemen pemroses (Processing Elements) akan bekerjasama dalam menyelesaikan sebuah masalah besar. Arsitektur paralel diperlukan karena kebanyakan mikroprosesor sekarang ini mempunyai fasilitas untuk mendukung multiprosesor; server dan workstation berarsitektur multiprosesor: Sun, SGI, DEC, COMPAQ; dan ikoprosesor yad (dan sekarang) adalah multiprosesor.

Untuk melakukan berbagai jenis komputasi paralel diperlukan infrastruktur mesin paralel yang terdiri dari banyak komputer yang dihubungkan dengan jaringan dan mampu bekerja secara paralel untuk menyelesaikan satu masalah. Untuk digunakan perangkat lunak pendukung yang biasa disebut middleware yang berperan mengatur distribusi antar titik dalam satu mesin paralel. Selanjutnya pemakai harus membuat pemrograman paralel untuk merealisasikan komputasi.

Yang perlu diingat adalah komputasi paralel berbeda dengan multitasking. Pengertian multitasking adalah komputer dengan processor tunggal mengeksekusi beberapa tugas secara bersamaan. Walaupun beberapa orang yang bergelut di bidang sistem operasi beranggapan bahwa komputer tunggal tidak bisa melakukan beberapa pekerjaan sekaligus, melainkan proses penjadwalan yang berlakukan pada sistem operasi membuat komputer seperti mengerjakan tugas secara bersamaan. Sedangkan komputasi paralel sudah dijelaskan sebelumnya, bahwa komputasi paralel menggunakan beberapa processor atau komputer. Selain itu komputasi paralel tidak menggunakan arsitektur Von Neumann.

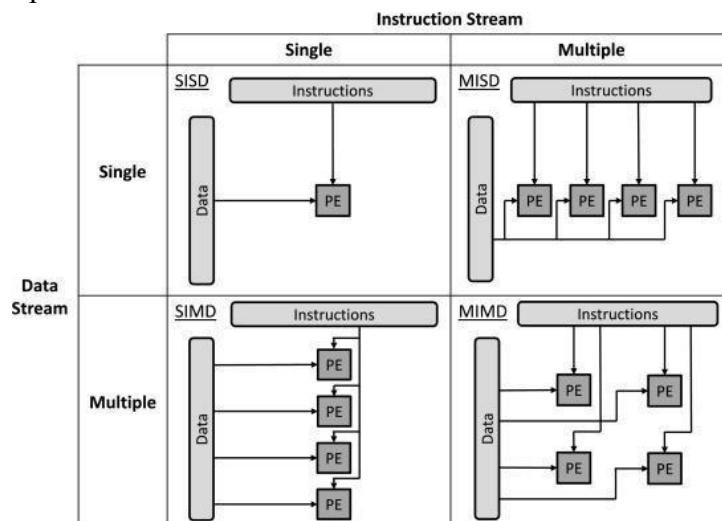
Berikut merupakan bagan arsitektur komputasi paralel yang akan dibahas pada makalah ini.



Gambar 2 Bagan Processor Organization

Sumber: <https://www.sciencedirect.com/topics/computer-science/>

Skema klasifikasi yang paling umum digunakan adalah taksonomi Flynn. Michael J. Flynn (1966) memperkenalkan suatu skema untuk mengklasifikasikan arsitektur suatu komputer dengan melihat bagaimana mesinnya menghubungkan instruksi-instruksinya ke data yang sedang diproses.



Gambar 3 Ilustrasi Sistem Taksonomi Flynn

Sumber

<https://www.sciencedirect.com/topics/computer-science/single-instructionsingle-data>

a. Single Instruction Single Data Stream

Arsitekur satu-satunya yang menggunakan arsitektur Von Neumann karena pada model ini hanya digunakan 1 processor saja dengan setiap instruksi dieksekusi

secara berurutan. Model ini bisa dikatakan sebagai model untuk komputasi tunggal karena keseluruhan mesin SISD memanfaatkan program *counter* yang melakukan sederetan eksekusi instruksi secara serial. Setiap masing-masing instruksi diambil dari memori, program counter akan selalu diupdate untuk mengisi alamat instruksi yang akan diambil dan dieksekusikan secara berurutan. Beberapa contoh komputer yang menggunakan model SISD adalah UNIVAC1, IBM 360, CDC 7600, Cray 1 dan PDP.

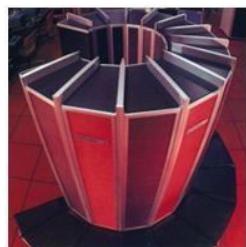


UNIVAC1
Source: <https://hpc.llnl.gov/>



CDC 7600
Source: <https://www.computer-history.info/>

Example of SISD



Cray 1
Source:
www.computerhistory.org/



PDP
Source: history-computer.com/pdp-1/



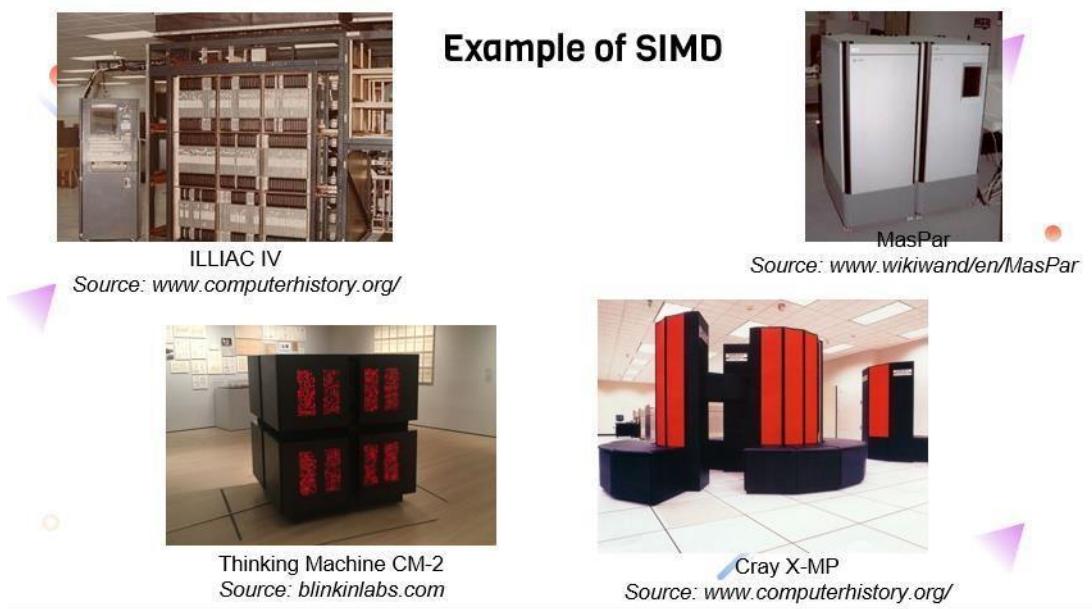
IBM 360
Source: <https://mnactec.cat/>

Gambar 4. Contoh Komputer dengan Model SISD

Kelebihan arsitektur SISD adalah membutuhkan lebih sedikit daya, dan tidak ada masalah protokol komunikasi yang kompleks di antara *multiple cores*. Sedangkan kekurangan arsitektur SISD adalah kecepatan arsitektur SISD terbatas karena hanya menggunakan *single-core processors* sehingga arsitektur ini tidak cocok untuk aplikasi yang lebih besar

b. Single Instruction Multiple Data Stream

SIMD menggunakan banyak processor dengan instruksi yang sama, namun setiap processor mengolah data yang berbeda. Sebagai contoh kita ingin mencari angka 27 pada deretan angka yang terdiri dari 100 angka, dan kita menggunakan 5 processor. Pada setiap processor kita menggunakan algoritma atau perintah yang sama, namun data yang diproses berbeda. Misalnya processor 1 mengolah data dari deretan / urutan pertama hingga urutan ke 20, processor 2 mengolah data dari urutan 21 sampai urutan 40, begitu pun untuk processor-processor yang lain. Beberapa contoh komputer yang menggunakan model SIMD adalah ILLIAC IV, MasPar, Cray X-MP, Cray Y-MP, Thinking Machine CM-2 dan Cell Processor (GPU).



Gambar 5. Contoh Komputer dengan Model SIMD

Kelebihan dari arsitektur SIMD, yaitu satu instruksi dapat melakukan operasi yang sama pada beberapa elemen, sistem *throughput* (jumlah data yang benar-benar terkirim dalam satu waktu tertentu) dapat ditingkatkan dengan meningkatkan jumlah *cores* prosesor, dan kecepatan pemrosesan lebih tinggi daripada arsitektur SISD. Sedangkan kelemahan arsitektur SIMD adalah terdapat komunikasi yang kompleks antara jumlah *cores* prosesor dan biaya yang diperlukan lebih tinggi dari arsitektur SISD.

c. *Multiple Instruction Single Data Stream*

MISD menggunakan banyak processor dengan setiap processor menggunakan instruksi yang berbeda namun mengolah data yang sama. Hal ini merupakan kebalikan dari model SIMD. Untuk contoh, kita bisa menggunakan kasus yang sama pada contoh model SIMD namun cara penyelesaian yang berbeda. Pada MISD jika pada komputer pertama, kedua, ketiga, keempat dan kelima sama-sama mengolah data dari urutan 1-100, namun algoritma yang digunakan untuk teknik pencariannya berbeda di setiap processor. Sampai saat ini belum ada komputer yang menggunakan model MISD.

d. *Multiple Instruction Multiple Data Stream*

MIMD menggunakan banyak processor dengan setiap processor memiliki instruksi yang berbeda dan mengolah data yang berbeda. Namun banyak komputer yang menggunakan model MIMD juga memasukkan komponen untuk model SIMD. Beberapa komputer yang menggunakan model MIMD adalah IBM POWER5, HP/Compaq AlphaServer, Intel IA32, AMD Opteron, Cray XT3 dan IBM BG/L.



Gambar 6. Contoh komputer dengan Model MIMD

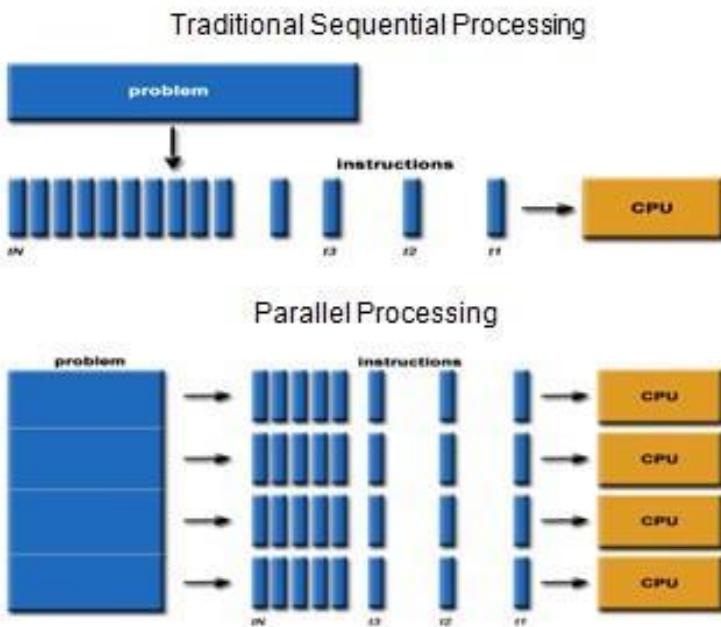
Komputer sekuensial berdasarkan klasifikasi Flynn adalah kelompok komputer SISD hanya mempunyai satu unit pengendali untuk menentukan instruksi yang akan dieksekusi. Pada setiap satuan waktu hanya satu instruksi yang dapat dieksekusi, dimana kecepatan akses ke memori dan kecepatan piranti masukan dan keluaran dapat memperlambat proses komputasi.

Beberapa metoda dibangun untuk menghindari masalah tersebut, seperti penggunaan cache memory. Namun komputer sekuensial ini tetap mengalami keterbatasan jika menangani masalah yang memerlukan kecepatan tinggi. Hal-hal tersebut di atas pada akhirnya melatarbelakangi lahirnya sistem komputer paralel. Berdasarkan klasifikasi Flynn, komputer paralel termasuk kelompok SIMD atau MIMD.

Komputer paralel mempunyai lebih dari satu unit pemroses dalam sebuah komputer yang sama. Hal yang membuat suatu komputer dengan banyak prosesor disebut sebagai komputer paralel adalah bahwa seluruh prosesor tersebut dapat beroperasi secara simultan. Jika tiap-tiap prosesor dapat mengerjakan satu juta operasi tiap detik, maka sepuluh prosesor dapat mengerjakan sepuluh juta operasi tiap detik, seratus prosesor akan dapat mengerjakan seratus juta operasi tiap detiknya.

Pada dasarnya aktivitas sebuah prosesor pada komputer paralel adalah sama dengan aktivitas sebuah prosesor pada komputer sekuensial. Tiap prosesor membaca (read) data dari memori, memprosesnya dan menuliskannya (write) kembali ke memori. Aktivitas komputasi ini dikerjakan oleh seluruh prosesor secara paralel.

Singkatnya untuk perbedaan antara komputasi tunggal dengan komputasi paralel, bisa digambarkan pada gambar di bawah ini:



Gambar 7. Ilustrasi Perbedaan Komputasi Tunggal dan Komputasi Paralel

Sumber: <https://www.dictio.id/t/apa-yang-dimaksud-dengan-parralel-processing/12267>

Dari perbedaan kedua gambar di atas, kita dapat menyimpulkan bahwa kinerja komputasi paralel lebih efektif dan dapat menghemat waktu untuk pemrosesan data yang banyak daripada komputasi tunggal.

Dari penjelasan-penjelasan di atas, kita bisa mendapatkan jawaban mengapa dan kapan kita perlu menggunakan komputasi paralel. Jawabannya adalah karena komputasi paralel jauh lebih menghemat waktu dan sangat efektif ketika kita harus mengolah data dalam jumlah yang besar. Namun keefektifan akan hilang ketika kita hanya mengolah data dalam jumlah yang kecil, karena data dengan jumlah kecil atau sedikit lebih efektif jika kita menggunakan komputasi tunggal.

Kita akan membahas pula dua skema lainnya yaitu: Shore dan Feng. J.E. Shore (dalam Soeparlan, 1995) membuat klasifikasi arsitektur komputer yang didasarkan pada organisasi bagian-bagian penyusun suatu komputer dan membedakannya menjadi enam jenis mesin.

- Mesin I. Pada komputer ini, satu instruksi dikerjakan pada suatu waktu dan masing-masing beroperasi pada satu word dalam suatu waktu.
- Mesin II. Komputer ini juga menjalankan satu instruksi pada suatu waktu, namun ia beroperasi pada sebuah irisan dari suatu bit dalam suatu waktu, bukannya semua bit dalam suatu word data.
- Mesin III. Sebuah komputer dalam kelas ini memiliki dua unit pengolahan yang dapat beroperasi pada data, satu word dalam suatu waktu atau suatu irisan bit dalam suatu waktu.

- d. Mesin IV. Komputer jenis ini dicirikan oleh sejumlah elemen (unit pengolahan dan unit memori), semua di bawah kendali sebuah unit kendali logika (CLU) tunggal.
- e. Mesin V. Mesin V dihasilkan dengan mengubah Mesin IV sedemikian sehingga elemenelemen pengolahan dapat berkomunikasi dengán tetangga terdekat mereka.
- f. Mesin VI. Komputer ini, disebut sebagai array logika-dalam-memori, merupakan sebuah mesin dengan logika prosesor yang tersebar dalam memori.

Tse-yum Feng (1972) menyarankan pengklasifikasian arsitektur komputer atas tingkatan paralelisme mereka. Tingkatan paralelisme (degree of parallelism) diwakili oleh pasangan (n, m) dimana n merupakan panjang word dan m adalah panjang irisan bit. Pasangan ini diklasifikasikan menjadi empat kelompok sebagai berikut:

- a. Jika $n = 1$ dan $m = 1$ maka tidak terjadi paralelisme. Word dan bit diproses satu per satuan waktu. Hal ini disebut sebagai word serial/bit serial(WSBS).
- b. Jika $n > 1$ dan $m = 1$ maka paralelisme itu disebut sebagai word paralel/bit serial (WPBS). Dalam hal ini, semua n irisan bit diproses satu per satuan waktu.
- c. Paralelisme word serial/bit paralel (WSBP) terjadi jika $n = 1$ dan $m > 1$. Dengan demikian sejumlah n word diproses satu per satuan waktu tetapi sejumlah m bit dan masing-masing word diproses secara paralel.
- d. Kategori terakhir disebut sebagai word paralel/bit paralel (WPBP) dan merupakan suatu paralelisme dimana $n > 1$ dan $m > 1$. Dalam hal ini, sejumlah nm bit diproses secara bersamaan.

3. Arsitektur Memori pada Komputasi Paralel

Pada komputer paralel, arsitektur memori diklasifikasikan menjadi tiga kategori antara lain:

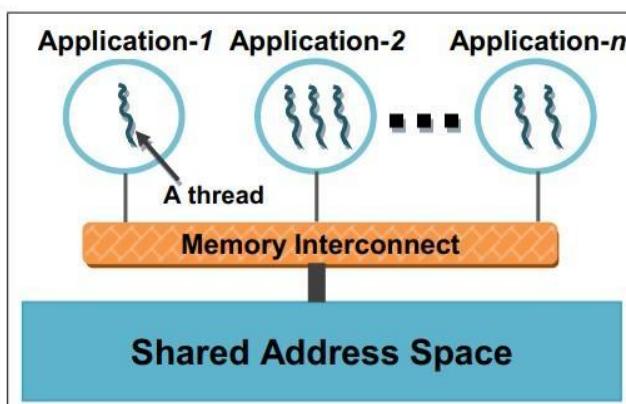
a. Shared memory

Memory dalam komputasi menurut BBC (n.a).adalah suatu area di mana data dan instruksi diingat. Dalam *memory system*, salah satu sistem yang banyak digunakan adalah *shared memory*. Menurut Rusling (n.a), *shared memory* adalah suatu sistem penyimpanan informasi di mana suatu informasi dalam memori bisa dipanggil kembali oleh beberapa program dalam waktu yang bersamaan. Dalam konteks prosesor, *shared memory* adalah bagian dari memori akses acak (RAM) yang dapat diakses oleh semua prosesor dalam sistem multi-prosesor. Pada arsitektur jenis ini, prosesor dapat mengakses semua memori sebagai space alamat global. *Shared memory* dibagi menjadi dua kelas yaitu UMA (Uniform Memory Access) dan NUMA (Non-Uniform Memory Access).

1) UMA (Uniform Memory Access)

Uniform Memory Access adalah sistem *memory* dalam *shared memory* yang memungkinkan prosesor untuk berbagi memori fisik yang sama (Ali & Syed, 2011). Karakteristik pada *shared memory UMA*, yaitu semua prosesor dapat mengakses memori sebagai ruang alamat bersama; setiap prosesor dapat bekerja secara *independent namun saling* berbagi memori; dan perubahan pada suatu lokasi memori oleh sebuah prosesor dapat diketahui oleh prosesor lain.

Berikut merupakan ilustrasi dari *Shared Memory UMA*.



Gambar 8. Ilustrasi Sistem *Shared Memory UMA*

Sumber: Aleem, 2015

Menurut Ali & Syed (2011), ada tiga tipe UMA, yaitu

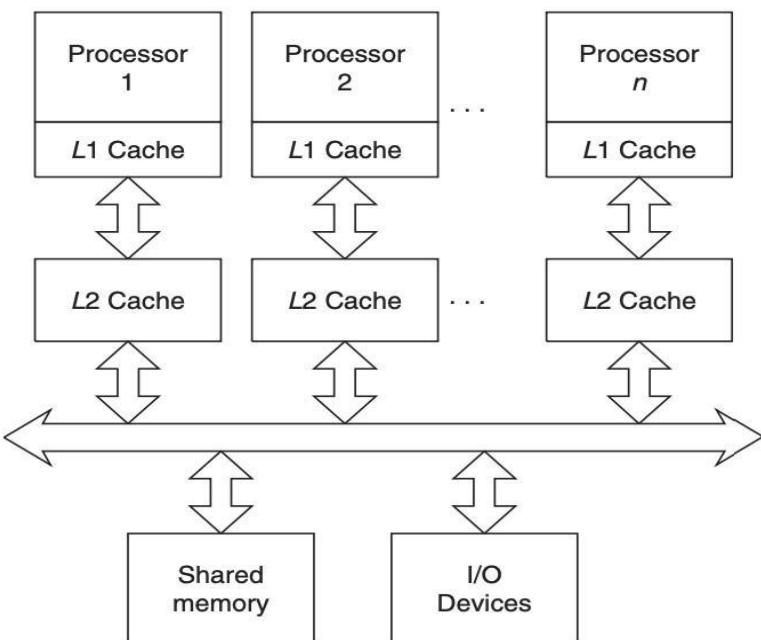
a) *Bus-symmetric multiprocessing architecture UMA*

Symmetric multiprocessing memungkinkan dua atau lebih prosesor identik terhubung ke satu memori utama bersama (memori utama ini disebut *bus*). Sistem ini memiliki akses penuh ke semua perangkat input dan output dan dikendalikan oleh sistem operasi tunggal yang memperlakukan semua prosesor secara setara (Patterson, 2018). Sistem ini adalah sistem yang paling populer dari semua sistem *symmetric multiprocessing* dan biasanya tersedia mulai dari server dengan hanya dua prosesor untuk hingga prosesor untuk sistem grafis berperforma tinggi, seperti Silicon Graphics's Power Challenge yang berisi hingga 36 prosesor (Cheung, 2005). Ciri-ciri sistem ini adalah sebagai berikut (Patterson dan Hennessy, 1998):

- i) Dua atau lebih prosesor yang identik diterapkan dalam sistem yang berdiri sendiri.
- ii) Semua prosesor berbagi memori dan perangkat *input output* yang sama melalui satu atau lebih *bus* bersama dengan waktu akses yang sama.
- iii) Semua prosesor mampu melakukan fungsi yang sama.

- iv) Distribusi beban kerja antara prosesor dilakukan oleh sistem operasi sedemikian rupa sehingga beberapa tugas independen atau dependen dibagi antara prosesor tanpa pertimbangan khusus dalam program aplikasi.

Distribusi beban kerja merupakan salah satu sistem ini banyak diadopsi dalam produk komersial. Kebanyakan sistem operasi modern, seperti MS-WindowsNT, Linux, dan Solaris, mendukung sistem ini. Pengguna tidak lagi perlu mempelajari keterampilan pemrograman paralel khusus untuk mengeksplorasi kinerja *symmetric multiprocessing*. Pengguna juga dapat memilih untuk menulis aplikasi *multithreaded* untuk mengeksplorasi kemampuan paralel dari *symmetric multiprocessing* dalam satu tugas. (Cheung, 2005)

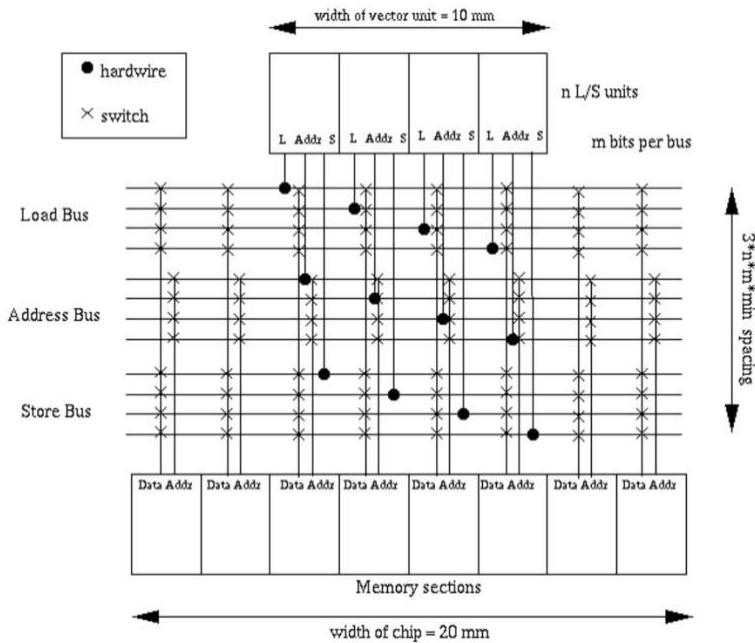


Gambar 9. Ilustrasi *symmetric multiprocessing* menurut Cheung (2005)

b) Crossbar section UMA

Dalam sistem ini, menurut Berkeley University (n.a), beberapa unit *load/store* (L/S) dari prosesor dihubungkan ke beberapa bagian memori. Unit L/S mampu mengeluarkan beban atau penyimpanan pada setiap siklus, dan sebagai hasilnya, bagian memori idealnya mampu memproses permintaan ini pada kecepatan yang sama. Dengan kata lain, jika ada n L/S unit dengan masing-masing mengeluarkan satu akses memori per siklus, maka memori harus memiliki bandwidth puncak minimal n kata per siklus.

Sistem *crossbar section* memungkinkan transfer simultan dari semua modul memori karena ada jalur terpisah yang terkait dengan setiap modul. Dengan demikian, perangkat keras yang diperlukan untuk mengimplementasikan sistem ini cukup besar dan kompleks.



Gambar 10. Ilustrasi crossbar section menurut Berkeley University

c) Multistage Interconnection Network (MIN)

Multistage interconnection network adalah kelas jaringan komputer berkecepatan tinggi yang biasanya terdiri dari elemen pemrosesan di satu ujung jaringan dan elemen memori di ujung lainnya dan dihubungkan oleh elemen *switching* (Nielsen, 2016). MIN biasanya digunakan dalam komputasi kinerja tinggi atau paralel sebagai interkoneksi latensi rendah.

MIN, menurut Nielsen (2016), dapat dikategorikan menjadi 3 tipe, yaitu:

i) Non-blocking

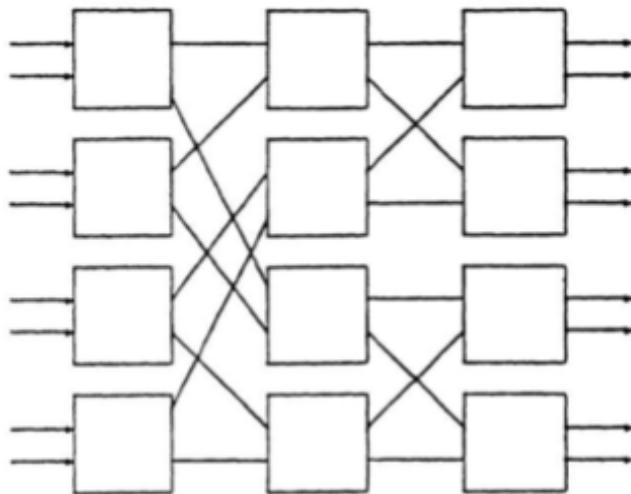
Jaringan non-blocking dapat menghubungkan input *idle* apapun ke output *idle* apa pun, terlepas dari koneksi yang sudah dibuat di seluruh jaringan.

ii) Rearrangable non blocking

Jenis jaringan ini dapat membuat semua kemungkinan koneksi antara input dan output dengan mengatur ulang koneksi yang ada.

iii) Blocking

Jenis jaringan ini tidak dapat mewujudkan semua kemungkinan koneksi antara input dan output. Ini karena koneksi antara satu input gratis ke output gratis lainnya diblokir oleh koneksi yang ada di jaringan.

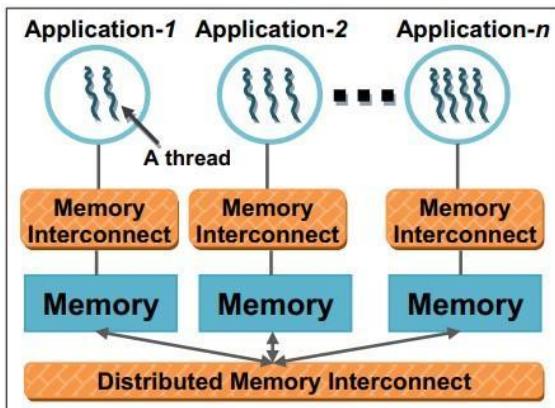


Gambar 11. Ilustrasi MIN oleh Chen (1991)

2) NUMA (Non-Uniform Memory Access)

Non-uniform memory access (NUMA) adalah desain memori komputer di mana waktu akses memori tergantung pada lokasi memori relatif terhadap prosesor. Dalam sistem ini, prosesor dapat mengakses memori lokalnya sendiri lebih cepat daripada memori non-lokal (memori lokal ke prosesor lain atau memori bersama antar prosesor). Manfaat NUMA terbatas pada beban kerja tertentu, terutama pada server di mana data sering dikaitkan dengan tugas atau pengguna tertentu (Marchanca & Anand, 2010).

Selanjutnya karakteristik pada *shared memory NUMA*, yaitu sejumlah prosesor memiliki bank alamat sendiri; prosesor dapat mengakses *memory local* dengan cepat, namun mengakses *memory remote* lebih lambat; seringkali dibuat dengan menggabungkan dua atau lebih prosesor SMP; dan satu prosesor SMP dapat mengakses memori prosesor lainnya secara langsung.



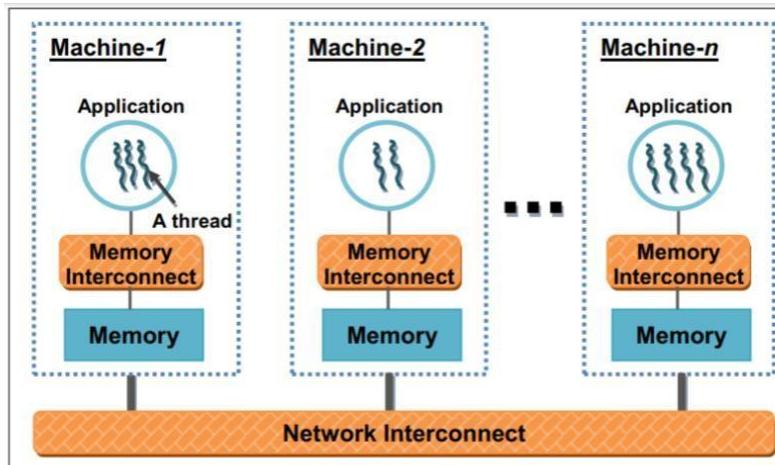
Gambar 12. Ilustrasi Sistem *Shared Memory NUMA*

Sumber: Aleem, 2015

Kelebihan menggunakan *shared memory* antara lain ruang alamat memori global memberikan kemudahan akses memori dari perspektif pemrograman dan proses berbagi data antar task lebih cepat dan seragam karena dekatnya memori ke CPU. Sedangkan kelemahan menggunakan *shared memory*, yaitu tidak scalable, artinya penambahan CPU dapat menambah trafik pada jalur *shared memory*; *programmer* bertanggung jawab untuk sinkronisasi yang memastikan akses yang tepat ke memori global; serta semakin bertambahnya prosesor maka semakin bertambah kompleks dan mahal.

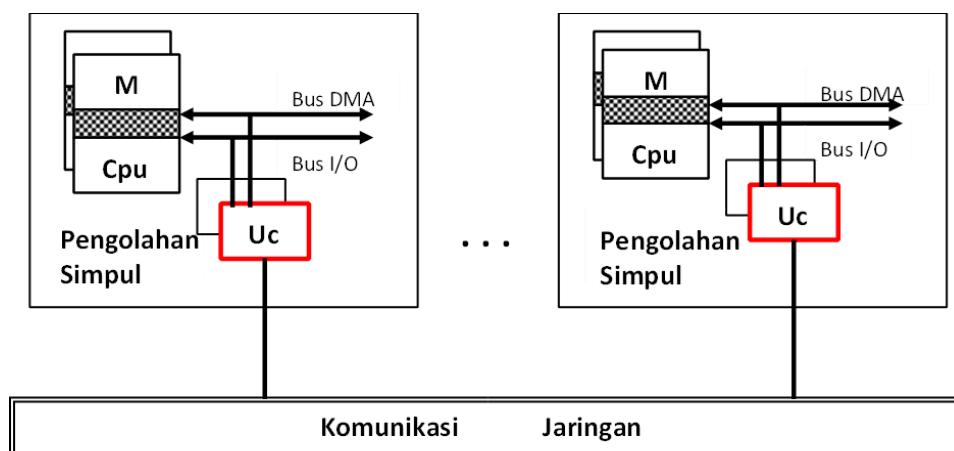
b. Distributed memory

Sistem memori terdistribusi membutuhkan jaringan komunikasi untuk menghubungkan memori antar prosesor. Prosesor pada arsitektur memori ini mempunyai memori lokal sendiri dan alamat memori dalam satu processor yang tidak terhubung untuk processor lain, sehingga tidak ada konsep ruang alamat global di semua processor. Perubahan pada memori lokal tidak mempengaruhi memori lain, oleh karena itu konsep koherensi cache tidak berlaku. Jika diperlukan pemrosesan interprosesor, tugas *programmer* menentukan bagaimana dan kapan data akan dikomunikasikan. Sinkronisasi antara tugas-tugas adalah juga tanggung jawab programer. Jaringan “fabric” digunakan untuk transfer data yang bervariasi, meskipun dapat sesederhana seperti Ethernet. Dalam *distributed memory*, node atau simpul pemrosesan tidak dapat berbagi ruang memori secara fisik dan tidak dapat mengakses memori node/simpul lainnya. *Distributed memory* memiliki I/O sebagai satusatunya mekanisme sederhana untuk kerjasama node atau simpul dengan pertukaran nilai eksplisit dan memungkinkan memori bersama dapat ditiru.



Gambar 13. Ilustrasi Sistem *Distributed Memory*

Sumber: Aleem, 2015



Gambar 14. Ilustrasi Unit I/O yang didedikasikan untuk node interfacing (UC)

Sumber: <http://groups.di.unipi.it/~vannesch/SPA/SPA-13-Cluster.pdf#>

Gambar 6. menunjukkan bahwa unit I/O yang didedikasikan untuk node atau simpul interfacing (UC) memiliki satuan komunikasi, satuan jaringan antarmuka, dan kartu jaringan. Distribusi memory memiliki beberapa jenis arsitektur memori, yaitu:

PC/Workstation Cluste, Multicuster, Massively Parallel Processor (MPP), Grid, Data Center, Server Farm, Cloud.

Kelebihan menggunakan distributed memory adalah *scalable*, yaitu jumlah prosesor dan jumlah memori dapat dengan mudah ditingkatkan secara proporsional, setiap prosesor cepat dan dapat mengakses memorinya sendiri tanpa interferensi atau campur tangan dan tanpa overhead yang dikeluarkan dengan berusaha mempertahankan cache koherensi global; dan *cost effective*, yaitu dapat menggunakan komputer komoditas, processor off-the-rak dan jaringan.

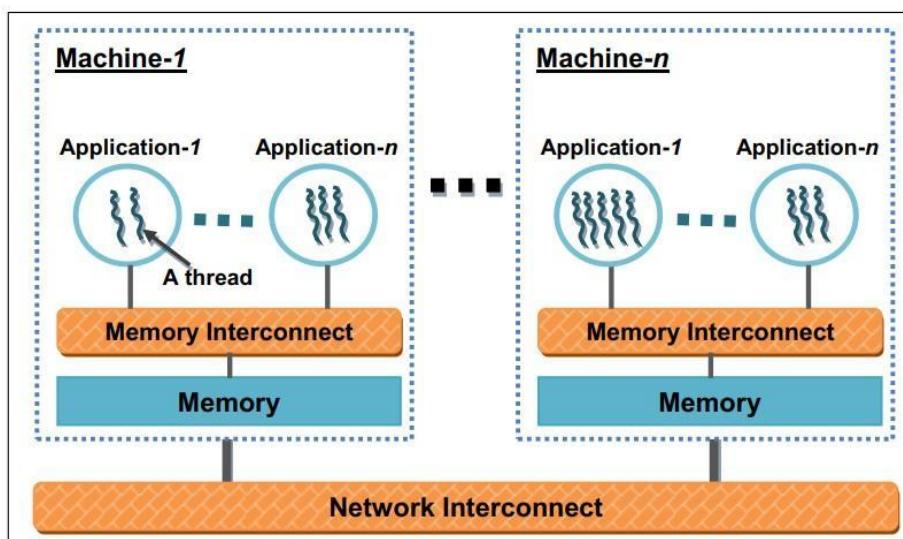
Sedangkan Kelemahan menggunakan *distributed memory* adalah tugas *programmer* semakin sulit terkait detail komunikasi data yang dimana programer bertanggung jawab untuk banyak rincian yang berkaitan dengan komunikasi data antar processor; sulit untuk memetakan struktur data yang ada berdasarkan memori global, non-seragam akses memori sehingga data yang berlaku pada remote simpul membutuhkan waktu yang lebih lama untuk mengakses data lokal di simpul.

c. Hybrid distributed-shared memory

Komputer terbesar dan tercepat di dunia saat ini mempekerjakan arsitektur *sharedmemory* dan *distributed-memory*. Arsitektur memori hybrid terdiri dari model komputasi yang berbeda dengan menggabungkan tipe shared dan distributed memory untuk membentuk hibrida. Komponen *shared memory* adalah komputer Symmetric Multiprocessing (SMP) koheren dimana prosesor memiliki akses global ke memori pada mesin tersebut. Sedangkan komponen *distributed memory* adalah jaringan komputer SMP multiple dimana setiap komputer SMP hanya mengenal memori miliknya saja. Komunikasi jaringan dibutuhkan untuk memindahkan data dari satu komputer SMP ke komputer SMP lainnya.

Model *hybrid memory* mengurangi komunikasi intra-node yang tidak perlu dan cocok untuk arsitektur komputasi berbasis multi prosesor. Setiap unit node adalah sistem *shared memory* dengan dua atau lebih *symmetric multiprocessors (SMPs)* yang membagikan sumber daya memory yang sama. Sistem distibutif dibentuk dengan menghubungkan node memori bersama melalui jaringan.

Berikut merupakan ilustrasi tren masa depan menggunakan arsitektur memori ini.



Gambar 15. Ilustrasi Sistem *Hybrid Memory*

Sumber: Aleem, 2015

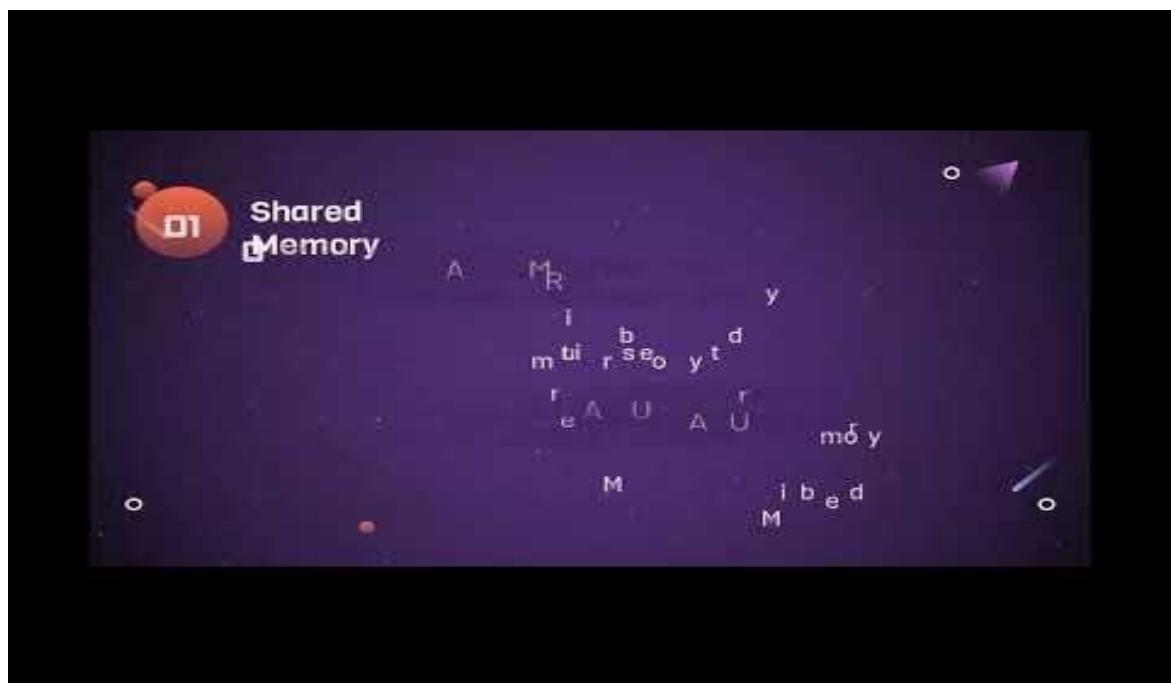
Gambar tersebut menunjukkan model *hybrid memory* dimana aplikasi paralel yang mengeksekusi menggunakan komunikasi. Keuntungan utama dari model hybrid adalah skalabilitas yang lebih baik (dibandingkan dengan model *shared-memory*) dan mengurangi latensi akses memori (dibandingkan dengan model *distributed-memory*).

PENUTUP

Kesimpulan yang dapat diambil dari makalah ini adalah sebagai berikut.

1. Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer secara bersamaan.
2. Dalam komputasi paralel, dikenal taksonomi Flynn, yaitu skema untuk mengklasifikasikan arsitektur suatu komputer dengan melihat bagaimana mesinnya menghubungkan instruksi-instruksinya ke data yang sedang diproses. Ada 4 skema menurut Flynn, yaitu *single instruction single data*, *single instruction multiple data*, *multiple instruction multiple data*, dan *multiple instruction single data*.
3. Pada komputer paralel, arsitektur memori diklasifikasikan menjadi tiga kategori yaitu *shared memory*, *hybrid memory*, dan *distributed memory*. *Shared memory* adalah bagian dari memori akses acak (RAM) yang dapat diakses oleh semua prosesor dalam sistem multi-prosesor. *Distributed memory* adalah arsitektur di mana prosesornya mempunyai memori lokal sendiri, sehingga inter-prosesor memori membutuhkan networking, sedangkan *hybrid memory* menggabungkan *shared* dan *distributed memory*.

Link video: [Arsitektur Komputer Paralel](#)



Daftar Pustaka

- Aleem, Muhammad. 2015. *A Java-based Programming and Execution Environment for Manycore Parallel Computers*. Proquest LLC is an Ann Arbor, Michigan, USA. diakses dari [ps://www.researchgate.net/publication/302956570_A_Java-based_Programming_an d_E_xecution_Environment_for_Many-core_Parallel_Computers](https://www.researchgate.net/publication/302956570_A_Java-based_Programming_and_Execution_Environment_for_Many-core_Parallel_Computers) tanggal 21 September 2022
- Ativ Mutsaqov, M. (n.d.). *MAKALAH SISTEM PARALEL DAN TERDISTRIBUSI “PARALLEL COMPUTING.”* diakses dari <https://pdfcoffee.com/tested-pdf-free.html> tanggal 21 september 2022
- Ali, Amjad & Syed, Khalid. 2011. *An Outlook of High Performance Computing Infrastructures for Scientific Computing*. Elsevier.
- BBC. *CPU and memory.* Diakses dari https://www.bbc.co.uk/bitesize/guides/zmb9mp3/revisi_on/6 tanggal 21 September 2022
- Chen, Kuo-Yu. 1991. *Multistage interconnection networks : improved routing algorithms and fault tolerance*. New Jersey Institute of Technology.
- Cheung, Peter Y.K. (2005). *The Electrical Engineering Handbook || Multiprocessors.*, Flynn, Michael J. 1966.. "Very high-speed computing systems". Proceedings of the IEEE.
- GeeksforGeeks. *Architecture of Distributed Shared Memory(DSM)* diakses dari <https://www.geeksforgeeks.org/architecture-of-distributed-shared-memorydsdm/> tanggal 21 September 2022

- GeeksforGeeks. *Computer Architecture | Flynn's taxonomy* diakses dari <https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/> tanggal 21 September 2022
- Golbus, dkk. *Interconnection Issues between Memory and Logic in IRAM Systems.* diakses dari <http://iram.cs.berkeley.edu/kozyraki/project/ee241/report/crossbar> tanggal 21 September 2022
- Hemmendinger, David .2016. "Computer memory". *Encyclopedia Britannica*. diakses dari <https://www.britannica.com/technology/computer-memory> tanggal 21 September 2022.
- Hofmaan, Frank. 2020. *Hardware: Understanding NUMA Architecture*. diakses dari https://linuxhint.com/understanding numa_architecture/ tanggal 21 September 2022
- HPCLLN. *Introduction to Parallel Computing Tutorial*. diakses dari https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computingtutorial#_Whatis tanggal 21 September 2022
- Nielsen, Frank (2016). "Topology of Interconnection Networks". *Introduction to HPC with MPI for Data Science*. Springer.
- Manchanda, Nakul & Anand, Karan. 2010. *Non-Uniform Memory Access (NUMA)*. New York University
- Patterson, David; Hennessy, John (2018). *Computer Organisation and Design: The Hardware/Software Interface* (RISC-V ed.). Cambridge
- Putra, Raden FAP. 2017. *Apa yang dimaksud dengan Parallel Processing?* diakses dari <https://www.dictio.id/t/apa-yang-dimaksud-dengan-parralel-processing/12267> tanggal 21 September 2022
- Rusling, David A. *Shared Memory*. Diakses dari <http://www.science.unitn.it/~fiorella/guidelin ux/tlk/node59.html> tanggal 21 September 2022
- Sammir, Haddad. 2015. 3. *Arsitektur Memori Komputer Paralel*. diakses dari <https://www.slideshare.net/hSammir/3-arsitektur-memori-komputer-paralel> tanggal 21 September 2022
- Soeparlan, Soepono. 1995. *Pengantar Organisasi Sistem Komputer: seri diktat kuliah*. Penerbit Gunadarma.
- ScienceDirect. *Single Instructon Single Data* diakses dari <https://www.sciencedirect.com/topics/computer-science/single-instruction-single-data> tanggal 22 September 2022
- Tutorialshand. *Flynn's classification of computers*. diakses dari <https://tutorialsinhand.com/tutorials/computer-organization-and-architecture-coatut orial/flynns-classification-of-computers/misd-architecture.aspx> tanggal 22 September 2022
- Tutorialspoint. *Parallel Computer Architecture Tutorial*. diakses dari https://www.tutorialspoint.com/parallel_computer_architecture/index.htm# tanggal 21 September 2022.

Tutorialspoint. *System and Memory Architecture*. Diakses dari
https://www.tutorialspoint.com/concurrency_in_python/concurrency_in_python_system_and_memory_architecture.htm tanggal 22 September 2022

Vanneschi, M. (n.d.). *Master Program (Laurea Magistrale) in Computer Science and Networking High Performance Computing Systems and Enabling Platforms 5. Distributed*

Memory Architectures. Diakses dari
<http://groups.di.unipi.it/~vannesch/SPA/SPA-13Cluster.pdf#> tanggal 21 september 2022

BAB 3

Studi Kasus Pemrograman Paralel dan Taksonomi Flynn

Cintya Kusuma Mahadhika, Elizabeth Lilies Megawati, Eka Dwi Agustina Ginting

Abstract

Telah dipelajari beberapa tipe pemrosesan dalam taksonomi Flynn. Dalam makalah ini, dibahas mengenai beberapa contoh kasus dalam taksonomi Flynn. Dijabarkan pula macam-macam alat pemrograman paralel dan studi kasus terkait. Dapat disimpulkan bahwa masing-masing pemrosesan maupun alat pemrograman memiliki algoritma maupun efisiensi masing-masing. Oleh karena itu, pembaca dapat menentukan alat pemrograman yang sesuai dengan masalah yang dimiliki.

Kata kunci: Taksonomy Flynn, GPU, alat pemrograman, komputasi paralel.

PENDAHULUAN

Seiring dengan perkembangan pemikiran manusia, teknologi kemudian berpaling dari perangkat analog menjadi perangkat digital yang multifungsi dan memiliki kehandalan yang tinggi. Pada dunia sains, komputasi memungkinkan penyelesaian problem yang tidak bisa diselesaikan melalui eksperimen tradisional maupun teoritis. Seiring dengan meningkatnya kebutuhan komputasi yang cepat dan naiknya harga peralatan komputer maka pengembangan aplikasi berbasis paralel termasuk komputasi paralel yang digunakan untuk menyelesaikan permasalahan waktu komputasi pada model komputasi standalone semakin banyak dilakukan. Pemanfaatan resource-resource komputasi yang tersedia secara paralel akan mempercepat eksekusi program. Program yang dieksekusi pada komputer master akan dibagi kepada node-node komputer untuk dimanfaatkan resourcennya. Komputasi ini sangat berguna untuk algoritma-algoritma yang memiliki tingkat pertumbuhan fungsi yang kuadratik dan eksponensial. Menggunakan konsep komputasi paralel, biaya investasi untuk komputer dapat ditekan dan konsumsi energi listrik, biaya instalasi serta pemeliharaan juga menjadi lebih rendah. Komputasi pararel fleksibel terhadap perubahan teknologi komputer yang sangat cepat. Alternatif populer saat ini adalah computer clustering (kelompok komputer) atau parallel computer (komputer paralel). Sistem ini merupakan penggabungan beberapa PC (disebut node) menjadi seolah-olah satu komputer dengan kemampuan yang lebih besar (Annas dkk, 2010).

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer independent. Hal ini umumnya diperlukan saat

kapasitas komputasi yang diperlukan sangat besar, baik karena pengolahan data maupun karena tuntutan proses komputasi yang banyak. Pada umumnya kasus ini ditemui yakni komputasi numerik yaitu menyelesaikan persamaan matematis di bidang fisika, kimia, teknik dan bidang lainnya. Dalam melakukan aneka jenis komputasi diperlukan infrastruktur mesin paralel yang terdiri dari komputer dan jaringan (perangkat lunak pendukung yang disebut middleware yang berperan dalam mengatur distribusi pekerjaan antar node dalam mesin paralel. Teknik pemrograman komputasi paralel adalah teknik eksekusi perintah atau operasi secara simultan baik dalam komputer satu prosesor ataupun dengan banyak prosesor (Annas dkk, 2010).

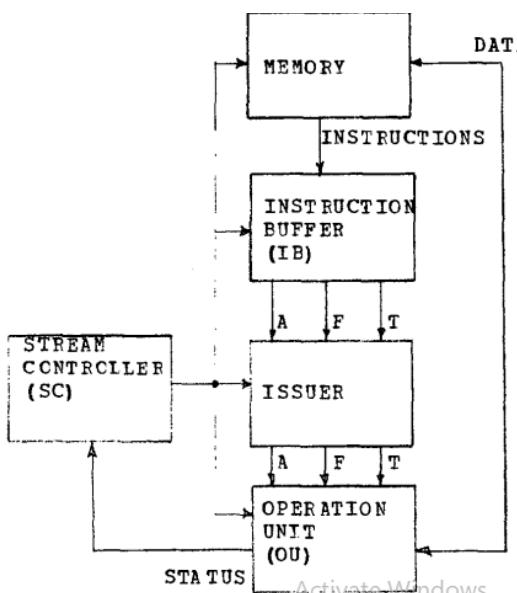
Dalam arsitektur komputer, taksonomi Flynn adalah klasifikasi yang membagi klasifikasi komputer dalam empat arsitektur, yaitu SISD (single instruction, single data), SIMD (single instruction, multiple data), MISD (multiple instruction, single data), dan MIMD (multiple instruction, multiple data) (Flynn, 1966). Masing-masing arsitektur memiliki ciri algoritma dan efisiensinya sendiri. Seperti halnya arsitektur komputer, pemrograman paralel juga memiliki tipe dan karakteristiknya sendiri. Dalam makalah ini, akan dibahas lebih lanjut mengenai studi kasus dalam taksonomi Flynn dan pemrograman paralel.

PEMBAHASAN

1. Studi Kasus Taksonomi Flynn

1. SISD (Single Instruction, Single Data)

Rosocha dan Lee (1979) melakukan penelitian berjudul “*Performance Enhancement of SISD Processor*”. Koordinasi otomatis eksekusi instruksi prosesor SISD diperiksa dalam konteks meminimalkan efek eksekusi cabang. Area, prefetch instruksi, resolusi cabang, dan organisasi penerbit diperiksa untuk kemungkinan perbaikan. Dorongan dari teknik ini adalah untuk memperluas pemrosesan bersyarat dan menggunakan pandangan ke depan untuk mendeteksi cabang. Buffer instruksi terstruktur, konvergen, digunakan untuk menyangga satu atau lebih level instruksi yang dibuat secara kondisional.



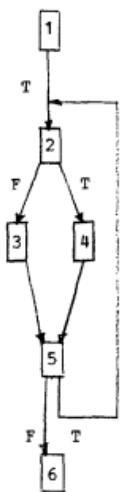
Gambar 1 kecepatan tinggi Processor SISD

```
Arithmetic/Logic
<opcode><dest. reg.>;<source1><source2>
(dest. reg = source1 opcode source2)

Memory Reference
<opcode><Data reg.><address reg>
(transmit data between data register and
memory location addressed by address
register.)

Branch
<opcode><branch address>
(Evaluate the predicate specified by
the opcode. If true, activate the
block at location <branch address>;
otherwise activate the immediate
successor block.)
```

Gambar 2 Sintaks instruksi



Gambar 3 struktur program statis

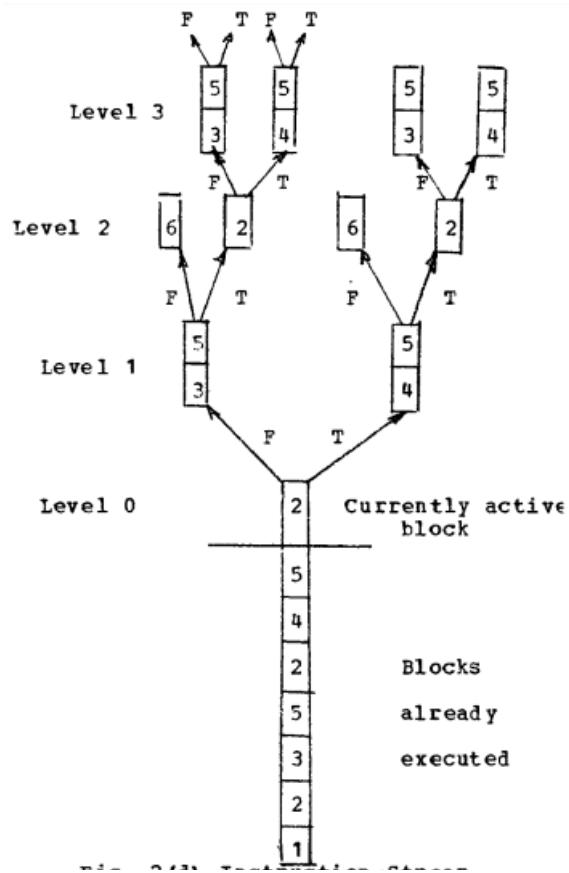
```

/*
 * Load the necessary address      */
/* registers, initialize loop    */
/* counter, index, and accumulator */
/* registers                      */
/*                                */
1. LD ABB; B
2. LD ABC; C
3. LD ABD; D
4. 'etc.,'

7. LD RN; N
8. LD RI; #0
9. LD RSUM; #0
*/
/*
 * BLOCK 2                      */
10. LOOP: INC RI
/* Fetch B(I)                   */
11. ADD BX; ABB, RI
12. LD R1; PX
/* Fetch C(I)                   */
13. ADD BX; ABC, RI
14. LD R2; PY
/* Generate A                   */
15. MUL R3; R1, R2
*/
/* Branch if A > or = 0        */
16. BGE POS
*/
/*
 * BLOCK 3                      */
17. ADD BX; ABC, RI
18. LD R1; PX
/* Generate D                   */
19. MUL R3; R3, R1
20. R SUM
*/
/*
 * BLOCK 4                      */
/* Fetch F(I)                   */
21. POS: ADD BY; ABC, RI
22. LD R2; PY
23. MUL R3; R3, R2
*/
/*
 * BLOCK 5                      */
/* Fetch G(I)                   */
24. SUM: ADD PX; ABC, RI
25. LD R2; RX
/* Generate new G(I)            */
26. ADD R3; RR, R3
27. ADD R3; R2, F3
28. ST R3; RX
29. ADD RSUM; RSUM, F3
/* Determine if looping completed */
30. DEC RN
31. BNE LOOP
*/
/*
 * BLOCK 6                      */
/* Store SUM                     */
32. ST RSUM; ABB

```

Gambar 4 kode mesin dari program contoh



Gambar 5 aliran instruksi

Konvergen meningkatkan instruksi penyediaan respon dengan memberikan yang tingkat yang lebih tinggi pada instruksi prefetch. Analisis menunjukkan bahwa terdapat manfaat mempertahankan lebih dari 1 tingkat prefetch. Selain itu, konvergen dapat menyediakan aliran instruksi bersyarat untuk organisasi yang ingin mengekstrak konkurensi instruksi tambahan secara dinamis yang dihambat oleh batas blok dinamis. Skema pengkondisian akan memungkinkan tumpang tindih resolusi cabang dengan eksekusi instruksi. Meskipun mobilitas CCI yang diharapkan kurang dari tiga instruksi, pengurangan waktu aktivasi akan dihemat pada sebagian kecil dari semua eksekusi cabang. Penghematan tambahan dimungkinkan dengan mengintegrasikan evaluasi predikat dalam unit fungsi. *Issuer organization* adalah yang mempromosikan tingkat penerbitan dan eksekusi yang tinggi. Keuntungan ini diperoleh dengan mendistribusikan fungsi masalah ke fase eksekusi. Pemrosesan bersyarat dimasukkan dengan memperluas algoritma Tomasulo. Yang jelas, area penting yang mempengaruhi kinerja prosesor SISD telah diabaikan. Ini termasuk akses operan, organisasi i/o, dan strategi reorganisasi

program. Namun, manfaat dari strategi organisasi yang disajikan di sini, respons yang cepat terhadap eksekusi cabang, tampaknya berguna secara umum.

2. SIMD (Single Instruction, Multiple Data)

SIMD menggambarkan komputer dengan beberapa elemen pemrosesan yang melakukan operasi yang sama pada beberapa titik data secara bersamaan (Stokes, 2000). Arsitektur SIMD mengeksplorasi paralelisme data dengan mengeluarkan instruksi yang sama ke semua ALU (Arithmetic Logical Units) di setiap siklus. Berikut ini akan diberikan contoh algoritma dalam SIMD seperti yang dicontohkan oleh Schmidt dkk (2018), yaitu algoritma perkalian matriks $A \times B = C$ (A matriks berukuran $m \times l$, B matriks berukuran $l \times n$, C matriks berukuran $m \times n$) menggunakan program C++ sebagai berikut.

```
1 #include <cstdint>      // uint32_t
2 #include <iostream>       // std::cout
3 #include <immintrin.h> // AVX intrinsics

void plain_tmm(float * A,
               float * B,
               float * C,
               uint64_t M,
               uint64_t L,
               uint64_t N) {

    for (uint64_t i = 0; i < M; i++) {
        for (uint64_t j = 0; j < N; j++) {
            float accum = float(0);
            for (uint64_t k = 0; k < L; k++)
                accum += A[i*L+k]*B[j*L+k];
            C[i*N+j] = accum;
        }
    }
}

void avx2_tmm(float * A,
              float * B,
              float * C,
              uint64_t M,
              uint64_t L,
              uint64_t N) {

    for (uint64_t i = 0; i < M; i++) {
        for (uint64_t j = 0; j < N; j++) {

            __m256 X = _mm256_setzero_ps();
            for (uint64_t k = 0; k < L; k += 8) {
                const __m256 AV = _mm256_load_ps(A+i*L+k);
                const __m256 BV = _mm256_load_ps(B+j*L+k);
                X = _mm256_fmadd_ps(AV,BV,X);
            }

            C[i*N+j] = hsum_avx(X);
        }
    }
}
```

Apabila kode tersebut dieksekusi dalam Intel i7-6800K CPU dengan dimensi matriks

$m = 1024, l = 2048, n = 4096$, maka akan diperoleh

elapsed time (avx2_tmm): 2.133s.

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

MISD adalah salah satu arsitektur dalam taksonomi Flynn di mana banyak unit fungsional melakukan operasi yang berbeda pada data yang sama (Flynn, 1966). MISD sangat jarang ditemukan dalam contoh nyata. Dalam beberapa literatur, disebutkan bahwa MISD digunakan dalam *systolic array*, yaitu proses di mana data input paralel mengalir melalui jaringan node prosesor terprogram, menyerupai otak manusia yang menggabungkan, memproses, menggabungkan atau mengurutkan data input menjadi hasil turunan (Flynn & Rudd, 1996). Namun, beberapa literatur juga menyebutkan bahwa *systolic array* tidak memenuhi klasifikasi MISD sebab input *systolic array* biasanya berupa vektor nilai independen serta nilai input ini digabungkan dan digabungkan ke dalam hasil dan tidak mempertahankan independensinya.

Kemudian, beberapa node tidak beroperasi pada data yang sama, yang membuat *systolic array* tidak memenuhi syarat untuk diklasifikasikan sebagai MISD. Alasan lain mengapa array sistolik tidak memenuhi syarat sebagai MISD adalah sama dengan alasan yang mendiskualifikasinya dari kategori SISD: data input biasanya berupa vektor, bukan nilai data tunggal, meskipun orang dapat berargumen bahwa vektor input apa pun yang diberikan adalah kumpulan data tunggal. (Quinn, 2004)

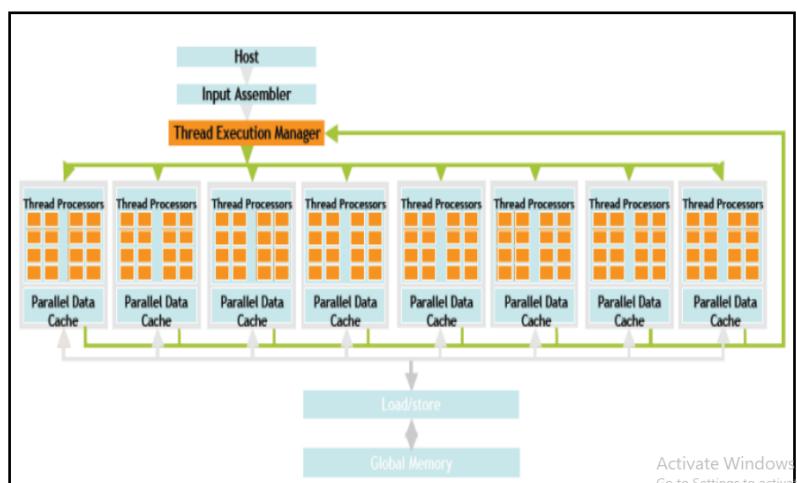
4. MIMD (*Multiple Instruction, Multiple Data*)

MIMD adalah suatu arsitektur komputer dalam komputasi Flynn di mana mesin memiliki sejumlah prosesor yang berfungsi secara asinkron dan independen. Setiap saat, prosesor yang berbeda dapat mengeksekusi instruksi yang berbeda pada bagian data yang berbeda (Flynn, 1966). Secara umum, ada dua tipe arsitektur MIMD, yaitu *true multiprocessor* dan *shared resource multiprocessor*. *Lockout time* atau waktu tunda prosesor dalam mengakses informasi dirumuskan sebagai berikut (Madnick, 2011).

$$L_j = \sum_i p_{ij} T_{ij}$$

5. GPU

GPU merupakan semacam prosesor yang ada di dalam kartu grafis yang didesain khusus untuk mengolah data gambar/ grafis agar dapat tampil ke monitor dengan cepat dan tepat. GPU memiliki kemampuan khusus, yaitu melakukan perhitungan matematika yang kompleks untuk membentuk gambar (geometri) dalam ruang 3-D, yang tidak dapat ditangani oleh CPU. (Nudirman, 2016). GPU memiliki arsitektur tertentu karena GPU merupakan processor *multithread* yang mampu mendukung jutaan pemrosesan data pada satu waktu. Berikut merupakan gambar ilustrasi arsitektur GPU.

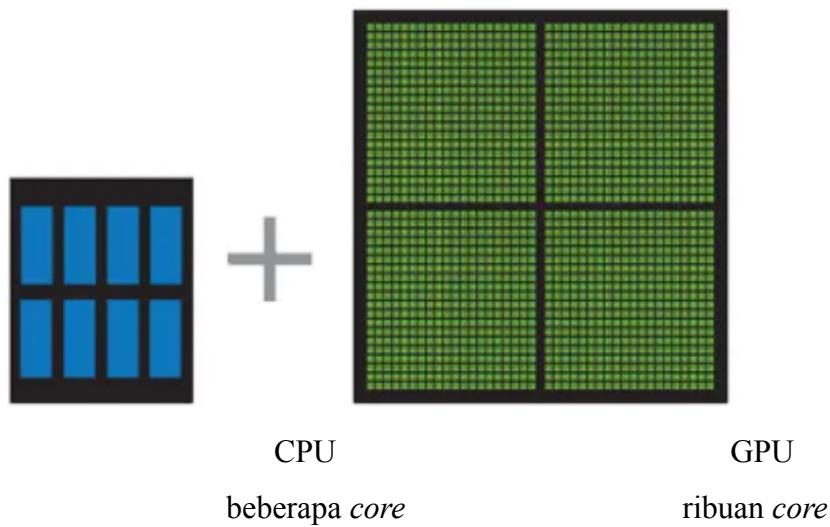


Gambar Arsitektur GPU

Sumber: Kurniawan, 2015

Sebagai ilustrasi, ketika kita bermain game, kemudian memainkan karakter game dari satu titik ke titik lain, atau melakukan sebuah aksi, pasti ruang, tata cahaya, warna, dan objek-objek di dalamnya berubah. Di balik itu, terdapat rumus-rumus matematika yang sangat kompleks yang memperhitungkan sistem koordinat, vektor, jarak, dan hukum-hukum fisika agar terbangun sebuah realitas tiga dimensi dan harus berjalan realtime. Begitupun dalam software editing, compositing, dan 3D yang membutuhkan formulasi yang sangat kompleks. (Nudirman, 2016)

Hal di atas hanya dapat dilakukan GPU bukan CPU, karena terdapat perbedaan arsitektur antara CPU dan GPU, yaitu CPU hanya memiliki beberapa inti prosesor (dual-core, quad-core, octa-core) yang bekerja secara serial, sedangkan GPU terdiri atas ribuan core kecil-kecil yang bekerja secara paralel. (Nudirman, 2016)



Gambar Ilustrasi Perbedaan banyak *Core* antara CPU dan GPU

Sumber: Nudirman, 2016

Tautan berikut merupakan ilustrasi untuk memahami perbedaan CPU dan GPU.
<https://youtu.be/-P28LKWTzrI>

6. Alat Pemrograman Paralel

1. OpenMP

OpenMP (Open MultiProcessing) merupakan sistem bahasa pemrograman paralel yang digunakan untuk mengekspresikan paralelisme *shared-memory*. (Mohr dkk,2001). Lebih lengkapnya, Open MP adalah *Application Program Interface (API)* untuk menulis aplikasi *multithreaded* (MT), berupa satu set direktif *compiler* dan *library* untuk pemrograman aplikasi paralel yang menyederhanakan penulisan program pada C++, C, Fortran. Multithreading adalah kemampuan CPU untuk menjalankan beberapa proses dalam waktu yang bersamaan, Kemampuan ini dapat mengurangi proses runtime dan meningkatkan efisiensi algoritma. Selain itu, *shared-memory* adalah arsitektur paralel yang memungkinkan setiap proses berjalan untuk mendapatkan sumber memori secara proporsional. Melalui fitur ini, waktu akses ke penyimpanan dapat berminimum dan kecepatan proses runtime dapat ditingkatkan. (Rabbani, 2018)

Contoh Studi Kasus

Rabbani (2018) melakukan penelitian berjudul Kinerja OpenMP pada Pengolahan Citra dengan Model Curvature Motion. Pada penelitian tersebut, solusi MCM diaproksimasi menggunakan skema *finite difference* dan disimulasikan ke dalam paralel OpenMP.

Hasil kinerja paralel dapat ditemukan pada tabel berikut.

Tabel Waktu eksekusi kinerja paralel algoritma FDM untuk gerak *curvature* menggunakan OpenMP

n_iterasi	Execution Time(s)			
	Serial	2 Threads	4 Threads	8 Threads
500	49.781	20.991	17.043	10.155
2000	102.671	43.275	22.872	20.918
4500	155.026	65.347	40.944	31.542
8000	204.490	86.235	45.822	41.966
12500	250.509	105.685	56.771	52.145
18000	292.458	123.952	67.051	61.764
24500	330.815	139.998	77.814	70.859
32000	365.567	155.080	86.817	79.253
40500	392.789	166.926	91.819	85.841
50000	409.380	180.422	156.002	93.302

Tabel Kecepatan dan efisiensi kinerja paralel algoritma FDM untuk gerak *curvature* menggunakan OpenMP

2 Threads	Speedup			Efficiency(%)		
	4 Threads	8 Threads	2 Threads	4 Threads	8 Threads	
2.372	2.921	4.902	119	73	61	
2.373	4.489	4.908	119	112	61	
2.372	3.786	4.915	119	95	61	
2.371	4.463	4.873	119	112	61	
2.370	4.413	4.804	119	110	60	
2.359	4.362	4.735	118	109	59	
2.363	4.251	4.669	118	106	58	
2.357	4.211	4.613	118	105	58	
2.353	4.278	4.576	118	107	57	
2.269	2.624	4.388	113	66	55	

Dari tabel tersebut dapat disimpulkan bahwa performa paralel pada pengolahan citra dengan model *Curvature Motion* membutuhkan waktu komputasi yang lebih rendah daripada algoritma serial. kode paralel dengan 2 *Thread* memiliki kecepatan rata-rata yang lebih rendah daripada menggunakan 4 *Thread* dan 8 *Thread*. Namun,rata-rata efisiensi kode paralel menggunakan 2 *Thread* lebih tinggi daripada menggunakan 4 *Thread* dan 8 *Thread*.

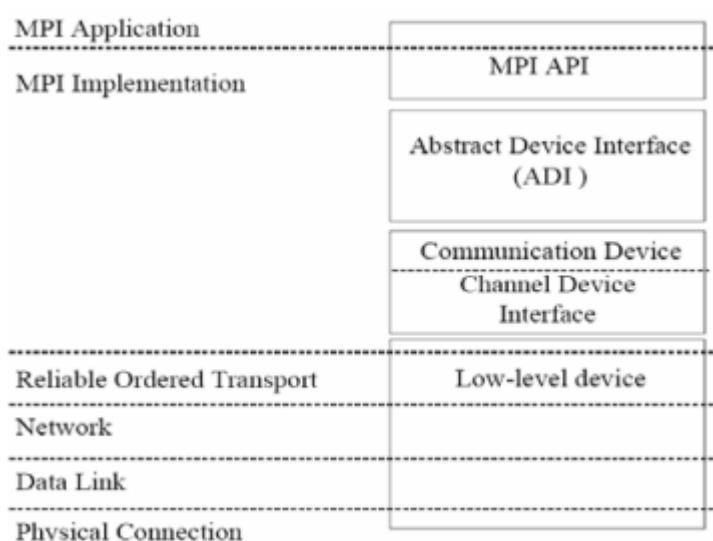
2. MPI

Komputasi paralel merupakan metode komputasi yang membagi beban komputasi ke dalam beberapa bagian kecil sub proses komputasi, dimana sub komputasi tersebut dijalankan pada *processor* yang berbeda secara bersamaan dan saling berinteraksi satu sama lain dalam menyelesaikan satu permasalahan komputasi. Salah satu protokol dalam pemrograman

paralel adalah MPI yang dikembangkan dalam skema *distributed memory*. (Wibawa dkk, 2018)

MPI (Message-Passing Interface) adalah sebuah standar untuk *interface* pada pemrograman paralel. MPI merupakan spesifikasi sekaligus library yang dikembangkan untuk mempermudah pembuatan program paralel, yang bertugas mengelola transfer data seluruh modul kalkulasi. MPI mengijinkan pertukaran data (*message*) antara *processor*. MPI dibuat oleh William Gropp, Ewing Lusk dan lainnya dan MPI diterima banyak komunitas pemrograman paralel karena sifatnya independen sehingga realisasinya beragam. (Annas, 2010)

Pada komputasi paralel, MPI ditujukan untuk SIMD dan MMID. MPI merupakan spesifikasi dan sekaligus library yang bisa dipanggil dan dideklarasikan sesuai kebutuhan. Sebelum bisa dipanggil dan dideklarasikan, spesifikasi MPI perlu di-*invoke* atau di-*include*. Jika diperlukan empat program client dan satu program server sehingga total diperlukan lima modul program, maka kelima-limanya juga harus melibatkan pemanggilan MPI sehingga transfer data antar modul bisa dilakukan. MPI memungkinkan banyak proses dieksekusi secara paralel pada waktu yang sama. Modul harus dikompilasi menggunakan kompilator yang sesuai agar dapat dieksekusi dalam arsitektur paralel. Apabila program cukup besar, maka perlu dibuat kompilasi otomatis menggunakan standard Makefile untuk MPI. Cara tersebut memungkinkan program besar dari berbagai kompilator seolah disatukan dan memungkinkan untuk bekerja sama dalam sebuah jaringan komputer atau dalam sistem komputer paralel. (Yustina, 2014).



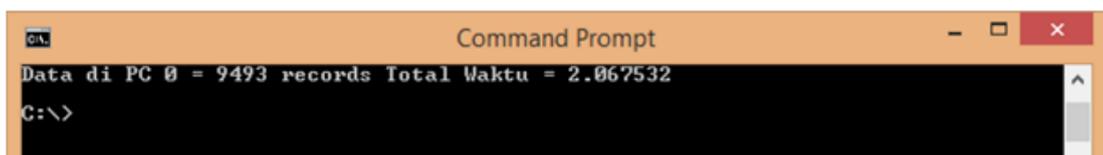
Gambar Arsitektur MPI versi terbaru (MPICH)

Sumber Suhartanto, 2006

Contoh Studi Kasus 1

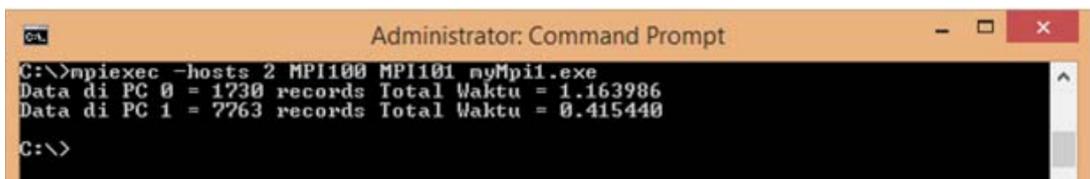
Wibawa (2018) melakukan penelitian tentang komputasi paralel menggunakan model *message passing* pada SIM RS (Sistem Informasi Manajemen Rumah Sakit). Dengan menggunakan MPI, diharapkan kemampuan komputasi paralel meningkat sehingga mampu menjadi solusi bagi permasalahan beban data dan kecepatan waktu komputasi yang terjadi pada SIM RS. Penelitian dilakukan dengan berfokus pada proses pencarian data manajemen rumah sakit.

Data SIMRS diolah menggunakan program paralel MPI dengan topologi jaringan paralel yaitu menggunakan 1 komputer/ CPU berfungsi sebagai master dan 6 komputer/CPU sebagai *slave* dengan total 7 komputer/CPU baik master ataupun *slave* dapat berfungsi mengolah data.



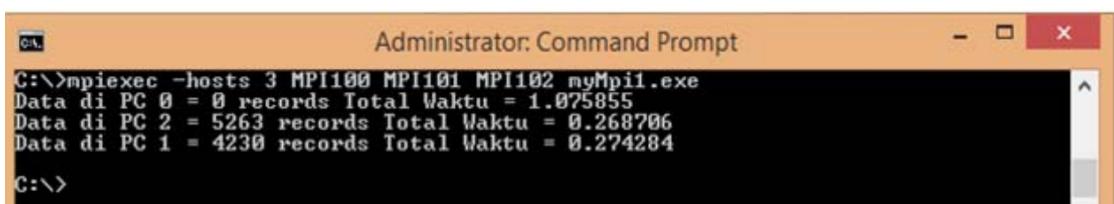
```
Command Prompt
Data di PC 0 = 9493 records Total Waktu = 2.067532
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 1 CPU



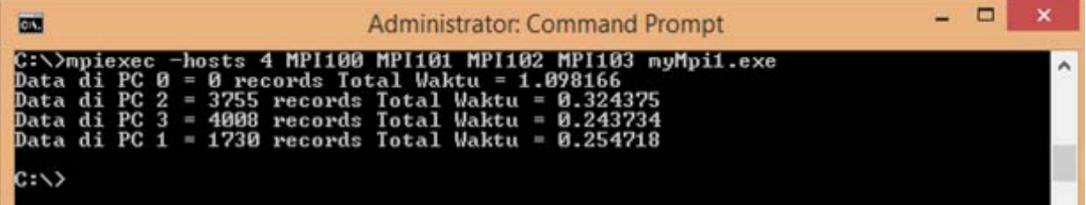
```
Administrator: Command Prompt
C:\>mpiexec -hosts 2 MPI100 MPI101 myMpi1.exe
Data di PC 0 = 1730 records Total Waktu = 1.163986
Data di PC 1 = 7763 records Total Waktu = 0.415440
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 2 CPU



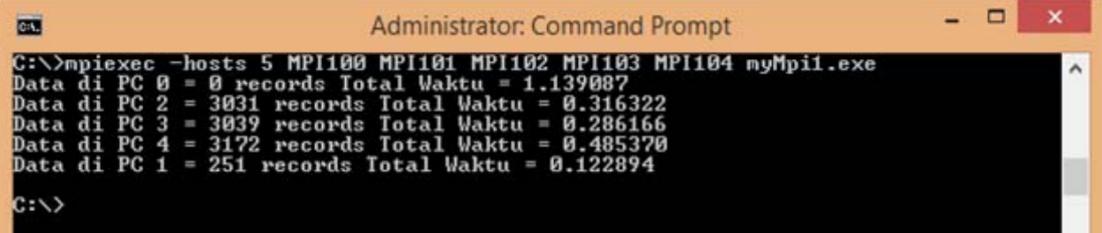
```
Administrator: Command Prompt
C:\>mpiexec -hosts 3 MPI100 MPI101 MPI102 myMpi1.exe
Data di PC 0 = 0 records Total Waktu = 1.075855
Data di PC 2 = 5263 records Total Waktu = 0.268706
Data di PC 1 = 4230 records Total Waktu = 0.274284
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 3 CPU



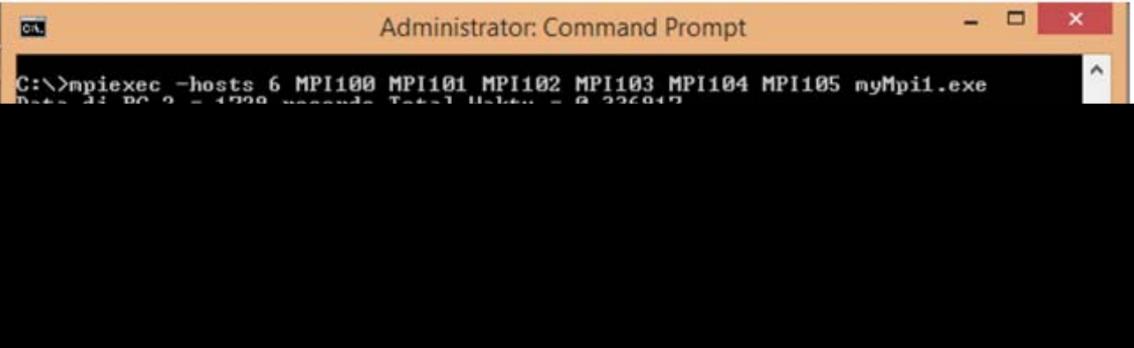
```
C:\>mpiexec -hosts 4 MPI100 MPI101 MPI102 MPI103 myMpi1.exe
Data di PC 0 = 0 records Total Waktu = 1.098166
Data di PC 2 = 3755 records Total Waktu = 0.324375
Data di PC 3 = 4008 records Total Waktu = 0.243734
Data di PC 1 = 1730 records Total Waktu = 0.254718
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 4 CPU



```
C:\>mpiexec -hosts 5 MPI100 MPI101 MPI102 MPI103 MPI104 myMpi1.exe
Data di PC 0 = 0 records Total Waktu = 1.139087
Data di PC 2 = 3031 records Total Waktu = 0.316322
Data di PC 3 = 3039 records Total Waktu = 0.286166
Data di PC 4 = 3172 records Total Waktu = 0.485370
Data di PC 1 = 251 records Total Waktu = 0.122894
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 5 CPU



```
C:\>mpiexec -hosts 6 MPI100 MPI101 MPI102 MPI103 MPI104 MPI105 myMpi1.exe
Data di PC 0 = 4729 records Total Waktu = 0.328472
Data di PC 1 = 4729 records Total Waktu = 0.328472
Data di PC 2 = 4729 records Total Waktu = 0.328472
Data di PC 3 = 4729 records Total Waktu = 0.328472
Data di PC 4 = 4729 records Total Waktu = 0.328472
Data di PC 5 = 4729 records Total Waktu = 0.328472
C:\>
```

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 6 CPU

Gambar Hasil program MPI pada topologi jaringan paralel menggunakan 7 CPU

Dari gambar-gambar tersebut, diperoleh waktu yang dibutuhkan dalam proses eksekusi pada topologi jaringan:

1. sekuensial = 2,067532 detik
2. paralel dengan 2 CPU = 1.163986 detik
3. paralel dengan 3 CPU = 1.075855 detik
4. paralel dengan 4 CPU = 1.098166 detik
5. paralel dengan 5 CPU = 1.139087 detik

6. paralel dengan 6 CPU = 1.163986 detik

7. paralel dengan 7 CPU = 1.224581 detik

sehingga dapat disimpulkan bahwa waktu pengolahan data pasien menggunakan program paralel MPI lebih cepat dibandingkan pengolahan data menggunakan topologi jaringan sekuensial/1 komputer/CPU.

Wibawa (2018) juga memperhitungkan nilai speed up dengan cara waktu hasil pengolahan data topologi jaringan sekuensial (2, 067532 detik) dibagi waktu pengolahan data menggunakan program paralel MPI sesuai dengan jumlah komputer/CPU yang digunakan.

Tabel Hasil Pengujian Speed Up

Tabel tersebut menunjukkan adanya peningkatan kecepatan sampai pada penggunaan komputasi paralel pada 3 CPU, namun pada pengujian menggunakan 4,5,6, dan 7 CPU, *speed up* sudah mulai menurun. Penurunan kecepatan sangat mungkin terjadi berdasarkan Hukum Amdahl yang tercantum dalam buku Kurniawan, A (2010) tentang Pemrograman Paralel MPI & C, bahwa semakin banyak prosesor maka kecepatan akan sampai pada titik jenuh. Secara keseluruhan pengujian *speedup* pada penelitian Wibawa (2018) jika dibandingkan dengan hasil pengolahan data menggunakan topologi jaringan sekuensial, pengolahan data menggunakan topologi jaringan paralel menggunakan 7 CPU terbukti memiliki kecepatan yang lebih baik.

Setelah itu, dilakukan pula perhitungan efisiensi yang merupakan perbandingan antara *speed up* dan jumlah CPU yang digunakan.

Tabel Hasil Pengujian Efisiensi

Penurunan nilai efisiensi yang terjadi pada setiap peningkatan jumlah CPU disebabkan karena adanya CPU yang tidak bekerja (*idle*). Proses komunikasi yang kompleks menyebabkan terjadinya kondisi dimana CPU tidak bekerja dan menunggu untuk mendapat perintah. Hasil nilai efisiensi menunjukkan semakin banyak penggunaan CPU maka semakin menurun nilai efisiensi.

Contoh Studi Kasus 2

(Annas dkk, 2010) melakukan penelitian tentang studi komputasi dan implementasinya pada matriks besar dengan memanfaatkan perangkat lunak seperti Coding (IDE) dan compliler C, MPI library, SSH (Secure Shell), Octave dan BCCD (Bootable cluster)

dengan memanfaatkan perangkat lunak seperti Coding (IDE) dan compiler C, MPI library, SSH (Secure Shell), Octave dan BCCD (Bootable cluster). Dalam penelitian ini digunakan tiga kasus penyelesaian aljabar linear dengan ordo diatas 1000, yaitu (1). Invers Matriks; (2). Penyelesaian Eigenvalue; dan (3). Penyelesaian persamaan linear simultan $Ax = b$. Pada tiga kasus penyelesaian digunakan matriks yang dibangkitkan secara random. Metode yang digunakan untuk menyelesaikan kasus invers matriks dan $Ax=b$ yakni metode gauss jordan dan untuk penyelesaian eigenvalue menggunakan metode power method.

Langkah yang dilakukan dalam penelitian ini adalah (1). Mempelajari algoritma untuk setiap kasus yang diberikan, melakukan perhitungan menggunakan Octave, mendeklarasikan data dan variabel yang dibutuhkan. (2). Menulis program untuk kasus yang diberikan dalam dalam bahasa C dan menyimpannya sebagai proses kompilasi A. (3). Mengujicoba program C apakah sudah tepat dan menghasilkan output yang benar dengan pembanding hasil perhitungan dengan menggunakan Octave. (4). Apabila sudah benar, maka tahap selanjutnya memodifikasi program kompilasi A dalam bahasa C dengan tambahan library MPI dan menyimpannya sebagai proses kompilasi B. (5). Waktu komputasi untuk setiap kasus dicatat untuk membuktikan bahwa sistem paralel dapat mempercepat proses komputasi dibandingan dengan komputasi standalone.

Perangkat penelitian yang digunakan pada penelitian ini berupa 4 buah komputer, KVM, Switch, beserta Kabel Jaringan disusun, seperti:

Gambar susunn perangkat keras paralel komputing.

Sumber: Annas dkk, 2010.

Kemudian dilakukan pengujian node yang bertujuan untuk mengetahui apakah node yang terkoneksi dalam sistem paralel pada network bejalan atau tidak, dengan cara mengirimkan pesan secara berantai antar rank, mulai dari rank 0 hingga kembali ke rank 0.

Gambar Aliran proses pengiriman pesan antar rank pada sistem paralel

Sumber: Annas dkk, 2010

Fungsi yang digunakan dalam pengujian ini adalah MPI Send pada pengiriman dan MPI Recv pada sisi penerima pesan. Hasil dari pengujian dapat dilihat pada gambar screen

shoot. Hasil pengujian menunjukkan bahwa setiap rank dapat saling berkomunikasi yang ditunjukkan dengan adanya tanda paket berhasil diterima dan rank aktif. Hal ini berarti bahwa sistem parallel sudah siap digunakan untuk menyelesaikan kasus numerik dengan matrik besar dengan komputasi parallel.

Gambar Screen shoot hasil uji node

Sumber: Annas dkk, 2010

Pada pengujian kasus, invers matrik diselesaikan menggunakan metode gauss jordan dimana setiap iterasi sebanyak N ordo matriks dihitung dengan menggunakan baris pivot yang telah dikirimkan ke masing-masing rank. Dari pengujian terhadap program standalone dan program MPI.

Gambar waktu komputasi untuk penyelesaian invers matrik

Sumber: Annas dkk, 2010

Pengujian Kasus Matriks Eigenvalu, nilai eigenvalue dihitung menggunakan metode power method dengan normalisasi kuadrat dari tiap baris matriks.

Gambar waktu komputasi penghitungan nilai eigenvalue

Sumber: Annas dkk, 2010

Pengujian Kasus $Ax=b$. metode yang digunakan sama seperti pada kasus pertama yakni Gauss Jordan, namun data yang dikirimkan kembali oleh rank merupakan vektor bukan berupa matriks.

Gambar waktu komputasi penyelesaian $Ax=b$

Sumber: Annas dkk, 2010

Program paralel untuk semua kasus pada penelitian ini berkonsep sama yakni memecah banyak data matriks yang kemudian disebar ke masing-masing node dan melakukan operasi aritmatika di tiap-tiap node. Suatu program paralel tidak serta merta dapat menghasilkan speedup yang tinggi, ada beberapa bagian serial dari program yang tidak bisa dipecah dalam bentuk paralel.

3. CUDA/ OpenCL

Terdapat suatu teknik yang diaplikasikan oleh para programer untuk mengoptimalkan GPU agar perangkat lunaknya dapat berjalan lebih cepat. Tidak semua kartu grafis dapat melakukannya, melainkan hanya dapat dilakukan oleh kartu-kartu grafis yang memiliki fitur CUDA (Compute Unified Device Architectur) atau OpenCL (Open Computing Language). Dengan kartu grafis yang memiliki GPU acceleration, para programmer dapat mengaplikasikan pemrograman dengan teknik paralel computing ke dalam core-core GPU (bersama-sama dengan CPU) sehingga proses-proses di dalam software menjadi lebih cepat.

Gambar Proses GPU Acceleration

Sumber: Nudirman, 2016

Salah satu GPU yang umum telah digunakan adalah Compute Unified Architectur (CUDA), yaitu salah satu contoh komputasi yang memanfaatkan GPU dengan arsitektur yang dikembangkan oleh NVIDIA (Nudirman, 2016). CUDA memungkinkan pengembang perangkat lunak membuat program yang berjalan pada GPU buatan NVIDIA dengan *syntax* yang mirip dengan *syntax* C. Arsitektur CUDA terdiri dari tiga bagian dasar yang membantu pengembang sistem memanfaatkan kemampuan komputasi secara efisien dari kartu grafis tersebut. CUDA membagi *device* ke *grid*, *block*, dan *thread* dalam struktur hierarki. Terdapat sejumlah *thread* dalam satu *block*, sejumlah *blok* dalam satu *grid* dan sejumlah *grid* dalam satu GPU, membuat proses paralel dapat dicapai dengan menggunakan arsitektur hierarkis yang sangat besar.

Perbedaan antara CUDA dan OpenCL, yaitu CUDA hanya dapat *disupport* dengan Nvidia, sedangkan OpenCL bersifat *open source*. Meskipun demikian, Nvidia juga *support* dengan openCL. Sehingga beberapa pengguna yang menggunakan Nvidia akan memiliki dua opsi, yaitu CUDA dan OpenCL bergantung dari serinya.

Contoh Studi Kasus pada CUDA vs OpenCV

Kurniawan (2015) melakukan pengujian komputasi GPU dengan CUDA dan Komputasi CPU untuk mengolah gambar secara paralel. Gambar yang dibandingkan adalah gambar yang sama dengan resolusi gambar yang bertahap, yaitu 1280x768, 1900x1200, 4000x2500, 7500x5000 *pixel* dengan masing-masing resolusi diwakilkan 10 gambar. Fungsi dalam algoritma yang digunakan sama, namun cara mengolah informasi berbeda. Pemrograman yang digunakan untuk komputasi CPU adalah OpenCV, sedangkan

untuk komputasi GPU menggunakan CUDA. Perangkat keras yang digunakan adalah *notebook* dengan processor Intel Core i7-3520M 2.90Ghz, RAM 4GB, sistem Operasi Microsoft Window 7 32 bit, dan Open CV versi 2.4.3. Untuk pemrograman CUDA menggunakan VGA Nvidia NVS 5400M 2GB RAM DDR3.

Tahapan algoritma dengan pemrograman CUDA yang dilakukan adalah sebagai berikut.

1. Memuat gambar
2. Mengalokasikan CPU *memory*
3. Mengalokasikan GPU *memory* dengan jumlah memory yang sama dengan CPU menggunakan fungsi *library* CudaMalloc.
4. Mengambil input data (data gambar/ image) dari CPU *memory* yang disebut dengan *host memory*
5. Menyalin data ke dalam GPU *memory* menggunakan fungsi *library* CudaMemcpy dengan parameter CudaMemcpyHostToDevice, sehingga data dari *host memory* akan diolah ke dalam *device memory* (GPU)
6. Melakukan pemrosesan dalam GPU *memory* di dalam *kernel* yakni untuk menjalankan komputasi paralel sesuai dengan *grid*, *block*, dan *thread* yang dibutuhkan untuk proses paralel
7. Menyakin data hasil (*result data*) dalam CPU *memory* menggunakan fungsi *library* CudaMemcpy dengan parameter CudaMemcpyDeviceToHost. Data hasil dari *device memory* (GPU) dikembalikan ke *host memory*(CPU)
8. Membebaskan GPU *memory* atau *thread* lain menggunakan fungsi *library* CudaFree.

Dari tahapan tersebut, kemudian disusun suatu fungsi yang akan dijalankan pada GPU dengan menentukan berapa *grid* dan *block* untuk menjalankan pengolahan gambar secara paralel.

```
1. extern "C" void cuda_parallel( unsigned char* h_in, unsigned char* h_out, int width, int height, int widthStep, int widthStepOutput, int channels){  
2.     unsigned char* d_in;  
3.     unsigned char* d_out;  
4.     cudaMalloc((void**)&d_in, width*height*channels);  
5.     cudaMalloc((void**)&d_out, width*height);  
6.     cudaMemcpy(d_in, h_in, width*height*channels*sizeof(unsigned char), cudaMemcpyHostToDevice);
```

```

7. dim3 block (16,16);
8. dim3 grid (width/16, height/16);
9. kernel_grayscale<<>>(d_in, d_out, width, height, widthStep, widthStepOutput,
channels);
10. cudaMemcpy(h_out, d_out, width*height*sizeof( unsigned char),
cudaMemcpyDeviceToHost); 11. cudaFree(d_in); cudaFree(d_out);}

```

Gambar Algoritma Fungsi CUDA

Setelah itu, hasil gambar yang diolah di dalam *device memory* (GPU) dan dikembalikan ke ke *host memory* (CPU) akan ditampilkan di dalam canvas, sehingga untuk *load* dan *save* gambar dilakukan oleh *host memory* di dalam CPU.

Operasi yang diuji dalam proses pengolahan adalah *grayscale*, negatif, dan deteksi tepi dengan metode *sobel filter*. Dari pengujian tiap operasi terhadap 10 gambar, dapat dihitung rata-rata waktu proses dari masing-masing komputasi yang disajikan pada tabel berikut.

Tabel Perbandingan waktu rata-rata *image processing* operasi *grayscale*

Tabel Perbandingan waktu rata-rata *image processing* operasi negatif

Tabel Perbandingan waktu rata-rata *image processing* operasi deteksi tepi

Dari tabel-tabel tersebut, dapat disimpulkan bahwa waktu proses pada CUDA lebih unggul dibandingkan OpenCV.

Contoh Studi Kasus OpenCL (CPU vs GPU)

Arvin (2017) membuat pustaka yang dapat menghitung perkalian matriks menggunakan algoritma Strassen secara paralel baik matriks berukuran kecil maupun besar dengan menggunakan algoritma OpenCL. Salah satu jenis pengujian yang dilakukan adalah pengujian waktu komputasi algoritma perkalian matriks. Penelitian ini mengolaborasi mengolaborasikan dua penelitian lain sehingga menjadi CPU dan GPU dengan kasus khusus algoritma Strassen pada perkalian matriks dengan ukuran yang sama. Salah satu parameter yang uji yang digunakan adalah jenis *device* OpenCL (CPU vs GPU).

Berikut ini merupakan hasil perbandingan waktu komputasi antara algoritma Naif dengan algoritma Strassen pada *device* CPU dan GPU

Sumber: Arvin (2017)

Pada tabel tersebut, dapat disimpulkan bahwa *device CPU* jauh lebih cepat apabila digunakan untuk menghitung matriks berukuran kecil, sementara *device GPU* lebih cepat apabila digunakan untuk menghitung matriks berukuran besar (lebih dari 128x128).

9. Cara Menganalisa Kinerja Komputasi Paralel

Pengukuran kinerja komputasi paralel dilakukan dengan membandingkan waktu eksekusi antara program sekuensial dengan program paralel. (dalam Akbar, 2014) Perhitungan waktu dilakukan dengan memakai fungsi `omp_get_wtime()`. Jumlah iterasi dibuat bervariasi untuk melihat pengaruhnya terhadap kinerja program paralel. Setelah didapatkan waktu eksekusinya, langkah selanjutnya adalah menghitung *speedup* dan efisiensi program paralel.

Secara umum, *Speedup* dikategorikan ke dalam dua bentuk yaitu *absolute speedup* dan *relative speedup*. *Absolute speedup* mengukur peningkatan kinerja suatu program paralel dibandingkan terhadap program sekuensial yang memiliki waktu eksekusi paling singkat. Karena program sekuensial yang paling cepat untuk suatu masalah belum tentu merupakan program sekuensial yang paling cepat untuk masalah lain, maka sebagai program sekuensial diambil program aplikasi yang umum digunakan untuk memecahkan masalah. *Relative speedup* mengukur peningkatan kinerja yang diperoleh dengan menjalankan suatu program paralel pada sejumlah prosesor dibandingkan dengan program paralel yang sama yang dijalankan pada satu prosesor. Secara matematis, *relative speed up* didefinisikan sebagai berikut. (Wanasurya dkk, 2017)

$$S = \frac{T_s}{T_p}$$

S adalah berapa kali lebih cepat waktu eksekusi program paralel (T_p) dibandingkan dengan program sekuensial (T_s).

Menurut hukum Amdahl's (dalam Aziz, 2016), peningkatan perhitungan paralel dibatasi oleh bagian konstan dari sistem, yaitu bagian serial perangkat lunak yang tidak dapat ditingkatkan secara paralel dan akan mendominasi kinerja perangkat lunak tersebut. Berbagai hambatan pada perangkat keras juga mempengaruhi kinerja perhitungan paralel, diantaranya waktu komunikasi antar bagian komputer, penyeimbangan beban pada prosesor, proses I/O,

dan lain-lain. Peningkatan kecepatan menurut Hukum Amdahl untuk perangkat lunak ditunjukkan dengan :

$$S(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

$S(N)$ = peningkatan kecepatan dengan N CPU

P = fraksi perangkat lunak yang bersifat paralel

Dengan asumsi permasalahan yang digunakan pada perhitungan percepatan adalah sama. Peningkatan kualitas perangkat keras dapat meningkatkan kecepatan perangkat lunak, tetapi tidak akan melawan hukum Amdahl. Dengan kata lain, hukum Amdahl tetap berlaku, tetapi dengan posisi kurva yang berbeda. Semakin banyak CPU yang digunakan, tidak dapat mencapai percepatan yang ideal. Meskipun demikian, jumlah CPU yang lebih banyak akan lebih baik untuk menyelesaikan permasalahan yang besar dan rumit.

Efisiensi adalah ukuran seberapa besar waktu prosesor dipakai dengan baik, yaitu *speedup* (S) dibagi dengan jumlah prosesor (p). Efisiensi ideal adalah 100%, namun pada praktiknya antara 0-100%. (dalam Akbar, 2014)

$$E = \frac{S}{p}$$

Contoh Studi Kasus 1

Contoh skema pengujian kinerja kerja paralel dalam Rahmatullah (2001), yaitu melakukan perkalian matriks dengan dimensi 2000x2000 dengan nilai yang berbeda di dalamnya. Pengujian tersebut akan melibatkan teknik *message passing* yang melibatkan 3 *threads*.

Gambar Pengujian matriks 2000x2000

Pengujian pada perkalian matriks dimensi 2000x2000 menghasilkan kecepatan sekuensial 59.37 detik dan kecepatan paralel 19.83 detik. Hal tersebut menghasilkan simpulan bahwa sistem paralel yang dilakukan dapat melakukan perkalian matriks dimensi besar dengan waktu pengerjaan lebih cepat dibandingkan dengan sekuensial.

Contoh Studi Kasus 2

Rahmatullah (2001) juga melakukan pengujian lain, yaitu simulasi proses folder gambar. Terdapat sebuah gambar berukuran 512*512 memiliki noise dikarenakan nilai pixel yang tidak sesuai. Nilai pixel tersebut megatur warna merah, hijau, dan biru. Terdapat salah satu filter gambar yaitu center weighted average dimana dikenal juga filter teknik gaussian yaitu

merubah nilai sebuah pixel berdasarkan perkalian pixel sebelumnya dengan nilai yang ditentukan lalu dibagi dengan rata-rata nilai pengali tersebut. Teknik ini dapat berguna untuk mengurangi noise pada gambar. Pada pengujian ini pixel dikalikan dengan nilai pengali sebanyak 3x3 dengan nilai pengalinya:

Tabel Blok Nilai Pengali 3x3

$$\begin{array}{ccc} -1.25 & 0 & -1.25 \\ 0 & 10 & 0 \\ -1.25 & 0 & -1.25 \end{array}$$

Penjumlahan dari setiap perkalian dibagi dari penjumlahan nilai pengali dan disimpan di center blok. Pada proses paralel, gambar akan difilter dengan masing-masing prosesor diberi nilai pixel untuk diproses dan dilakukan gather untuk dijadikan satu gambar kembali.

Gambar yang Diuji

Gambar asli (kiri), dengan metode sekuensial (tengah), dengan metode paralel (kanan)

Pengujian skema ini membuktikan bahwa filter yang dibuat dapat mengurangi noise dari gambar original. Namun, akan terjadi perbedaan yang sangat mencolok apabila terjadi kesalahan perhitungan. Hal tersebut dapat terlihat dari hasil filter metode paralel yaitu terjadi perbedaan hasil perhitungan pixel sehingga membuat gambar menjadi terang. Berikut ini tabel parameter hasil pengujian gambar menggunakan metode sekuensial dan paralel, khusus untuk paralel digunakan skenario menggunakan 2, 4, 6, 8, dan 10 processor.

Sekuensial	Paralel	Speed Up	Efisiensi	Efisiensi
0.044868	0.023398	1.91759979	0.9587999	95.88%
	0.020579	2.18028087	0.54507022	54.51%
	4.83763	0.00927479	0.0015458	0.15%
	9.066256	0.0049489	0.00061861	0.06%
	11.177281	0.00401421	0.00040142	0.04%

Proses filter menunjukkan bahwa optimal waktu yang didapatkan adalah pada simulasi 4 prosesor. Hal ini dikarenakan penggunaan alat uji yaitu sebuah laptop dengan spesifikasi quadcore (4 prosesor). Terjadi perbedaan yang sangat mencolok bila dipaksakan melakukan kinerja dengan konfigurasi lebih dari 4 processor. Berdasarkan hal tersebut, dapat disimpulkan bahwa proses komputasi dapat berjalan maksimal sesuai dengan ketepatan

jumlah prosesor yang digunakan. Selanjutnya, lihat pada speed up dan efisiensi yang didapat. Pada kinerja paralel dengan 4 prosesor (sesuai skema pengujian ini), diperoleh nilai speed up p sekitar 2 kali lebih baik dibandingkan dengan sekuensial, dan nilai efisiensi yang dihasilkan sekitar 50.26% sehingga menunjukkan bahwa alat uji yang digunakan dapat melakukan proses komputasi yang lebih komplek.

Kesimpulan

1. SIMD menggambarkan komputer dengan beberapa elemen pemrosesan yang melakukan operasi yang sama pada beberapa titik data secara bersamaan dan arsitektur SIMD mengeksplorasi paralelisme data dengan mengeluarkan instruksi yang sama ke semua ALU (Arithmetic Logical Units) di setiap siklus.
2. MIMD adalah suatu arsitektur komputer dalam komputasi Flynn di mana mesin memiliki sejumlah prosesor yang berfungsi secara asinkron dan independen. Setiap saat, prosesor yang berbeda dapat mengeksekusi instruksi yang berbeda pada bagian data yang berbeda.
3. MISD adalah salah satu arsitektur dalam taksonomi Flynn di mana banyak unit fungsional melakukan operasi yang berbeda pada data yang sama dan MISD digunakan dalam *systolic array*, yaitu proses di mana data input paralel mengalir melalui jaringan node prosesor terprogram, menyerupai otak manusia yang menggabungkan, memproses, menggabungkan atau mengurutkan data input menjadi hasil turunan.
4. GPU memiliki arsitektur tertentu karena GPU merupakan processor *multithread* yang mampu mendukung jutaan pemrosesan data pada satu waktu. Salah satunya semacam prosesor yang ada di dalam kartu grafis yang didesain khusus untuk mengolah data gambar/ grafis agar dapat tampil ke monitor dengan cepat dan tepat. GPU memiliki kemampuan khusus, yaitu melakukan perhitungan matematika yang kompleks untuk membentuk gambar (geometri) dalam ruang 3-D, yang tidak dapat ditangani oleh CPU.
5. Alat yang digunakan dalam pemrograman paralel, yaitu: (1) OpenMP (Open Multi Processing) adalah *Application Program Interface (API)* untuk menulis aplikasi *multithreaded* (MT), berupa satu set direktif *compiler* dan *library* untuk pemrograman aplikasi paralel yang menyederhanakan penulisan pada program untuk mengurangi proses runtime dan meningkatkan efisiensi algoritma. (2) MPI (Message-Passing Interface) adalah sebuah standar untuk *interface* pada pemrograman paralel. MPI merupakan spesifikasi sekaligus library yang dikembangkan untuk mempermudah pembuatan program paralel, yang bertugas mengelola transfer data seluruh modul kalkulasi. MPI mengijinkan pertukaran data (*message*) antara *processor*. (3). GPU yang umum telah digunakan adalah Compute Unified Architectur (CUDA), yaitu salah satu contoh komputasi yang memanfaatkan GPU dengan arsitektur yang dikembangkan oleh NVIDIA. Arsitektur CUDA terdiri dari tiga bagian dasar yang membantu pengembangan sistem memanfaatkan kemampuan komputasi secara efisien dari kartu grafis tersebut. CUDA membagi *device* ke *grid*, *block*, dan *thread* dalam struktur hirarki. Terdapat sejumlah *thread* dalam satu *block*, sejumlah *blok* dalam satu *grid* dan sejumlah *grid*

- dalam satu GPU, membuat proses paralel dapat dicapai dengan menggunakan arsitektur hirarkis yang sangat besar.
6. Perbedaan antara CUDA dan OpenCL, yaitu CUDA hanya dapat *disupport* dengan Nvidia, sedangkan OpenCL bersifat *open source*. Meskipun demikian, Nvidia juga *support* dengan openCL. Sehingga beberapa pengguna yang menggunakan Nvidia akan memiliki dua opsi, yaitu CUDA dan OpenCL bergantung dari serinya.
 7. Analisis kerja komputasi paralel dilakukan dengan membandingkan waktu eksekusi antara program skuensial dengan program paralel. Perhitungan waku dilakukan dengan memakai fungsi. Jumlah iterasi dibuat bervariasi untuk melihat pengaruhnya terhadap kinerja program paralel. Setelah didapatkan waktu eksekusinya, langkah selanjutnya adalah meghitung *speedup* dan efisiensi program paralel.
 8. ***Speedup*** dikategorikan ke dalam dua bentuk yaitu *absolute speedup* dan *relative speedup*. *Absolute speedup* mengukur peningkatan kinerja suatu program paralel dibandingkan terhadap program sekuensial yang memiliki waktu eksekusi paling singkat. **Efisiensi** adalah ukuran seberapa besar waktu prosesor dipakai dengan baik dan efisiensi ideal adalah 100%.

Daftar Pustaka

Azis, Mohammad T, Azizul K, & Bintoro A. 2016. Kinerja Perhitungan Kritikalitas Mcnp6 Pada Komputasi Paralel. *Jurnal Seminar Keselamatan Nuklir*. diakses dari https://inis.iaea.org/collection/NCLCollectionStore/_Public/50/022/50022722.pdf tanggal 13 Oktober 2022

Akbar, Auriza R. 2014. *Pemrosesan Paralel*. diakses dari <https://adoc.pub/pemrosesan-paralel-auriza-rahmad-akbar.html> tanggal 12 Oktober 2022

Annas, Mayzar., I Gede, L.A. Syamsul Irfan. 2010. Studi Komputasi Paralel dan Implementasinya pada Kasus Komputasi Matriks Besar. diakses dari http://jcosine.if.unram.ac.id/public/journals/1/jCossin2016_PaperExp.pdf pada 12 oktober 2020

Arvin, dkk. 2017. Optimisasi Pustaka Untuk Perkalian Matriks Menggunakan Algoritma Strassen Berbasis Opencl. *Jurnal Seminar Nasional Inovasi Dan Aplikasi Teknologi Di Industri 2017*. diakses dari <https://ejournal.itn.ac.id/index.php/senjati/article/view/1591/141> 5 tanggal 11 Oktober 2022

Flynn, Michael J., Rudd, Kevin W. 1996. *Parallel Architectures*. CRC Press.

Flynn, Michael J. (December 1966). "Very high-speed computing systems". *Proceedings of the IEEE*. **54** (12): 1901–1909

Kurniawan, A. 2010. Pemrograman Paralel dengan MPI & C. ANDI Yogyakarta

Kurniawan, dkk. 2015. Analisis Perbandingan Komputasi GPU dengan CUDA dan Komputasi CPU untuk Image dan Video Processing. *Jurnal Seminar Nasional Aplikasi Teknologi Informasi (SNATi)*. diakses dari <http://etd.repository.ugm.ac.id/pemelitian/detail/117016> tanggal 11 Oktober 2022

Madnick, S.E. "Multi-processor software lockout," in Proc. concerning parallel processing and parallel processors," Proc. 1968 Ass. Comput. Mach. Nat. Conf., pp. 19-24.

Mohr, dkk. 2001. Design and Prototype of a Performance Tool Interface for OpenMP. *Journal of Supercomputing*. diakses dari https://www.researchgate.net/publication/2380669_Design_and_Prototype_of_a_Performance_Tool_Interface_for_OpenMP tanggal 11 Oktober 2022

Nudirman, Dani. 2016. Apa itu CUDA, GPU Acceleration, dan Mercury Playback Engine? diakses dari rumaheditor.com: <http://rumaheditor.com/apa-itu-cuda-gpu-acceleration-dan-mercury-playback-engine/> tanggal 11 Oktober 2022

Rahmatullah dkk. 2021. Analisis Perbandingan Performa Pemrograman Sekuensial dan Paralel Dengan Skema Uji Matrix, Filter Dan Quick Sort. *Jurnal Teknologi Informasi Universitas Lambung Mangkurat (JTIULM)* 6. 19 - 24. 10.20527/jtiulm.v6i1.69. diakses dari https://www.researchgate.net/publication/351300511_Analisis_Perbandingan_Performa_Pemrograman_Sekuensial_Dan_Paralel_Dengan_Skema_Uji_Matrix_Filter_Dan_Quick_Sort tanggal 11 Oktober 2022

Rosocha, W. G. & Lee E.S. 1979. Performance Enhancement of SISD Processors. *Journal of Computer Systems Research Group, University of Toronto, Canada*. diakses dari <https://dl.acm.org/doi/pdf/10.1145/800090.802912> tanggal 13 Oktober 2022

Schmidt, Bertil, et. Al. 2018. *Parallel Programming: Concept and Practice*. Morgan Kaufmann.

Stokes, Jon. 2000. *SIMD architecture*. diakses dari <https://arstechnica.com/features/2000/03/simd/> tanggal 13 Oktober 2022

Suhartanto, Heru. 2006. Kajian Perangkat Bantu Komputasi Paralel pada Jaringan PC. Makara, Teknologi, Vol. 10, no.2 November 2006. Diakses pada 12 oktober 2022 di

<https://media.neliti.com/media/publications/148472-ID-kajian-perangkatbantu-komputasi-parallel.pdf>

Quinn, Michael J. *Parallel Programming in C with MPI and OpenMP*. Boston: McGraw Hill, 2004.

Wanasurya, Andri L, Sri M, & Maria A K. 2017. Komputasi Paralel Untuk Pengolahan Prestasi Akademik Mahasiswa. *Jurnal Teknologi Elektro, Universitas Mercu Buana*. diakses dari <https://publikasi.mercubuana.ac.id/index.php/jte/article/view/2180> tanggal 13 Oktober 2022

Wibawa, dkk. 2018. Komputasi Paralel Menggunakan Model Message Passing Pada SIM RS (Sistem Informasi Manajemen Rumah Sakit). *Jurnal Majalah Ilmiah Teknologi Elektro*, Vol. 17, No. 3, September - Desember 2018. diakses dari <https://pdfs.semanticscholar.org/f12c/dd107d592d05dd58b95af73046c133930191.pdf> tanggal 11 Oktober 2022

Google Docs: (PPT dan Video)

https://univindonesia-my.sharepoint.com/:w/g/personal/elizabeth_lilies_office_ui_ac_id/EdSQ_xHLYiBJr8PL32ayaUsByppDTC4S7MRGFkw9jo9nGQ?e=K2vlAO

BAB 4

Aplikasi Parallel Computing (*Shared Memory, Distributed Memory, dan Hybrid Memory*)

Cintya Kusuma Mahadhika, Elizabeth Lilies Megawati, Eka Dwi Agustina Ginting

Abstrak

Kemajuan dari generasi yang semakin maju (*next-generation*) telah mengarah pada ketersediaan kumpulan data besar yang digunakan oleh berbagai aplikasi dalam berbagai bidang. Karena jumlah data menjadi luas dan besar (terstruktur, tidak terstruktur, semi terstruktur), maka menganalisis dan memproses data tersebut dalam berbagai aplikasi seperti sensor data, perawatan kesehatan, e-commerce membutuhkan teknologi pemrosesan big data. Metode komputasi yang marak digunakan dalam pemrosesan big data adalah komputasi paralel. Dalam makalah ini, dipaparkan definisi penggunaan memori dalam komputasi paralel (*shared memory, distributed memory, dan hybrid memory*) serta aplikasinya.

Kata kunci: *big data, shared memory, distributed memory, hybrid memory*

Pendahuluan

Kemajuan dari generasi yang semakin maju (*next-generation*) telah mengarah pada ketersediaan kumpulan data besar yang digunakan oleh berbagai aplikasi dalam berbagai bidang [Abui, 2020]. Karena jumlah data menjadi luas dan besar (terstruktur, tidak terstruktur, semi terstruktur), maka menganalisis dan memproses data tersebut dalam berbagai aplikasi seperti sensor data, perawatan kesehatan, e-commerce membutuhkan teknologi pemrosesan big data.

[Narayanan, 2020]

Big data adalah payung besar yang terdiri dari banyak aktivitas seperti menangkap, menyimpan, dan memproses data dalam jumlah besar. [Bassil, 2019] Dalam beberapa tahun terakhir, salah satu penelitian yang paling diminati adalah dalam optimalisasi kinerja sistem big data. Berbagai industri dan pemerintah memberikan penekanan pada teknologi big data, karena teknik komputasi konvensional yang tidak data memberikan hasil dan efisiensi yang diharapkan untuk mengelola data besar. [Rahman, 2021] Big data melibatkan kemampuan untuk mengelola sejumlah data yang berbeda, pada kecepatan yang tepat, dan dalam kerangka waktu yang tepat

untuk memungkinkan pemrosesan dan analisis data secara *real-time*. [Bassil, 2019]

Yang [in Fei Hui, 2018] mengungkapkan pemrosesan big data memerlukan strategi manajemen data yang efisien, algoritma komputasi paralel yang kompleks, dan sumber daya komputasi yang dapat diskalakan.

Pembahasan

1. Aplikasi *Shared Memory*

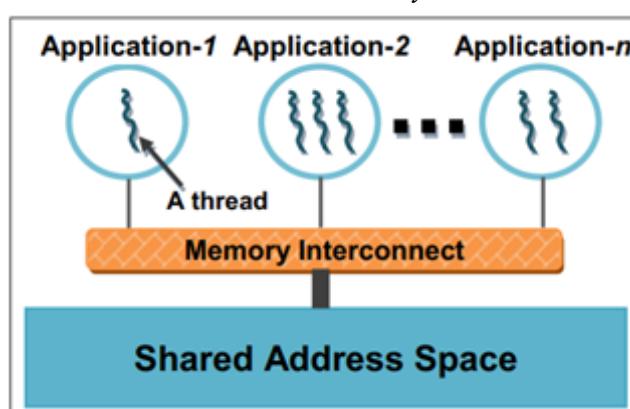
a. Definisi Shared Memory

Memory dalam komputasi menurut BBC (n.a) adalah suatu area di mana data dan instruksi diingat. Dalam *memory system*, salah satu sistem yang banyak digunakan adalah *shared memory*. Menurut Rusling (n.a), *shared memory* adalah suatu sistem penyimpanan informasi di mana suatu informasi dalam memori bisa dipanggil kembali oleh beberapa program dalam waktu yang bersamaan. Dalam konteks prosesor, *shared memory* adalah bagian dari memori akses acak (RAM) yang dapat diakses oleh semua prosesor dalam sistem multi-prosesor. Pada arsitektur jenis ini, prosesor dapat mengakses semua memori sebagai space alamat global. *Shared memory* dibagi menjadi dua kelas yaitu UMA (Uniform Memory Access) dan NUMA (Non-Uniform Memory Access).

1. UMA (Uniform Memory Access)

Uniform Memory Access adalah sistem *memory* dalam *shared memory* yang memungkinkan prosesor untuk berbagi memori fisik yang sama (Ali & Syed, 2011). Karakteristik pada *shared memory UMA*, yaitu semua prosesor dapat mengakses memori sebagai ruang alamat bersama; setiap prosesor dapat bekerja secara *independent namun saling* berbagi memori; dan perubahan pada suatu lokasi memori oleh sebuah prosesor dapat diketahui oleh prosesor lain.

Berikut merupakan ilustrasi dari *Shared Memory UMA*.



Gambar 8. Ilustrasi Sistem *Shared Memory UMA*

Sumber: Aleem, 2015

Menurut Ali & Syed (2011), ada tiga tipe UMA, yaitu:

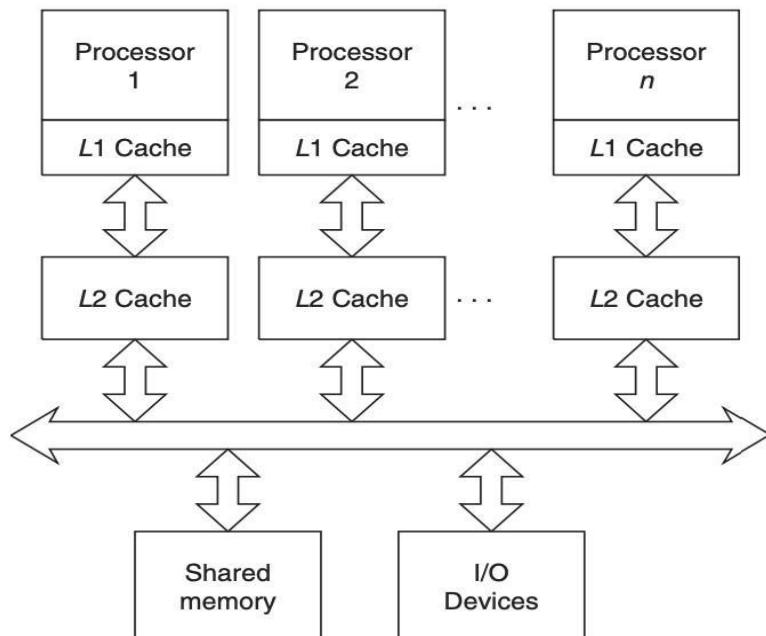
a. Bus-symmetric multiprocessing architecture UMA

Symmetric multiprocessing memungkinkan dua atau lebih prosesor identik terhubung ke satu memori utama bersama (memori utama ini disebut *bus*). Sistem

ini memiliki akses penuh ke semua perangkat input dan output dan dikendalikan oleh sistem operasi tunggal yang memperlakukan semua prosesor secara setara (Patterson, 2018). Sistem ini adalah sistem yang paling populer dari semua sistem *symmetric multiprocessing* dan biasanya tersedia mulai dari server dengan hanya dua prosesor untuk hingga prosesor untuk sistem grafis berperforma tinggi, seperti Silicon Graphics's Power Challenge yang berisi hingga 36 prosesor (Cheung, 2005). Ciri-ciri sistem ini adalah sebagai berikut (Patterson dan Hennessy, 1998):

- i. Dua atau lebih prosesor yang identik diterapkan dalam sistem yang berdiri sendiri.
- ii. Semua prosesor berbagi memori dan perangkat *input output* yang sama melalui satu atau lebih *bus* bersama dengan waktu akses yang sama.
- iii. Semua prosesor mampu melakukan fungsi yang sama.
- iv. Distribusi beban kerja antara prosesor dilakukan oleh sistem operasi sedemikian rupa sehingga beberapa tugas independen atau dependen dibagi antara prosesor tanpa pertimbangan khusus dalam program aplikasi.

Distribusi beban kerja merupakan salah satu sistem ini banyak diadopsi dalam produk komersial. Kebanyakan sistem operasi modern, seperti MS-WindowsNT, Linux, dan Solaris, mendukung sistem ini. Pengguna tidak lagi perlu mempelajari keterampilan pemrograman paralel khusus untuk mengeksplorasi kinerja *symmetric multiprocessing*. Pengguna juga dapat memilih untuk menulis aplikasi *multithreaded* untuk mengeksplorasi kemampuan paralel dari *symmetric multiprocessing* dalam satu tugas. (Cheung, 2005)



Gambar 9. Ilustrasi *symmetric multiprocessing* menurut Cheung (2005)

b. Crossbar section UMA

Dalam sistem ini, menurut Berkeley University (n.a), beberapa unit *load/store* (L/S) dari prosesor dihubungkan ke beberapa bagian memori. Unit L/S mampu mengeluarkan beban atau penyimpanan pada setiap siklus, dan sebagai hasilnya, bagian memori idealnya mampu memproses permintaan ini pada kecepatan yang sama. Dengan kata lain, jika ada n L/S unit dengan masing-masing mengeluarkan satu akses memori per siklus, maka memori harus memiliki bandwidth puncak minimal n kata per siklus. Sistem *crossbar section* memungkinkan transfer simultan dari semua modul memori karena ada jalur terpisah yang terkait dengan setiap modul. Dengan demikian, perangkat keras yang diperlukan untuk mengimplementasikan sistem ini cukup besar dan kompleks.

Gambar 10. Ilustrasi crossbar section menurut Berkeley University

c. Multistage Interconnection Network (MIN)

Multistage interconnection network adalah kelas jaringan komputer berkecepatan tinggi yang biasanya terdiri dari elemen pemrosesan di satu ujung jaringan dan elemen memori di ujung lainnya dan dihubungkan oleh elemen *switching* (Nielsen, 2016). MIN biasanya digunakan dalam komputasi kinerja tinggi atau paralel sebagai interkoneksi latensi rendah. MIN, menurut Nielsen (2016), dapat dikategorikan menjadi 3 tipe, yaitu:

i. Non-blocking

Jaringan non-blocking dapat menghubungkan input *idle* apapun ke output *idle* apa pun, terlepas dari koneksi yang sudah dibuat di seluruh jaringan.

ii. Rearrangable non blocking

Jenis jaringan ini dapat membuat semua kemungkinan koneksi antara input dan output dengan mengatur ulang koneksi yang ada.

iii. Blocking

Jenis jaringan ini tidak dapat mewujudkan semua kemungkinan koneksi antara input dan output. Ini karena koneksi antara satu input gratis ke output gratis lainnya diblokir oleh koneksi yang ada di jaringan.

Gambar 11. Ilustrasi MIN oleh Chen (1991)

2. NUMA (Non-Uniform Memory Access)

Non-uniform memory access (NUMA) adalah desain memori komputer di mana waktu akses memori tergantung pada lokasi memori relatif terhadap prosesor. Dalam sistem ini, prosesor dapat mengakses memori lokalnya sendiri lebih cepat daripada memori non-lokal (memori lokal ke prosesor lain atau memori bersama antar prosesor). Manfaat NUMA terbatas pada beban kerja tertentu, terutama pada server di mana data sering dikaitkan dengan tugas atau pengguna tertentu (Marchanca & Anand, 2010).

Selanjutnya karakteristik pada *shared memory NUMA*, yaitu sejumlah prosesor memiliki bank alamat sendiri; prosesor dapat mengakses *memory local* dengan cepat, namun mengakses *memory remote* lebih lambat; seringkali dibuat dengan menggabungkan dua atau lebih prosesor SMP; dan satu prosesor SMP dapat mengakses memori prosesor lainnya secara langsung.

Gambar 12. Ilustrasi Sistem *Shared Memory NUMA*

Sumber: Aleem, 2015

Kelebihan menggunakan *shared memory* antara lain ruang alamat memori global memberikan kemudahan akses memori dari perspektif pemrograman dan proses berbagi data antar task lebih cepat dan seragam karena dekatnya memori ke CPU. Sedangkan kelemahan menggunakan *shared memory*, yaitu tidak scalable, artinya penambahan CPU dapat menambah trafik pada jalur *shared memory*; programmer bertanggung jawab untuk sinkronisasi yang memastikan akses yang tepat ke memori global; serta semakin bertambahnya prosesor maka semakin bertambah kompleks dan mahal.

b. Aplikasi *Shared Memory*

Studi Kasus 1

Salah satu aplikasi *shared memory* adalah masalah OpenMP, misalnya dalam GAMESS (Bak et. al., 2021) GAMESS (*(General Atomic and Molecular Electronic Structure System)*) adalah paket perangkat lunak dengan berbagai struktur elektronik metode kimia kuantum, seperti Hartree–Fock (HF) dan teori gangguan Moller-Plesset orde kedua dengan resolusi pendekatan identitas. Sebagian besar kodennya di Fortran 77/90, dengan pustaka C/C++ opsional yang menggunakan CUDA untuk membongkar kode ke GPU NVIDIA. GAMESS secara tradisional menggunakan MPI bersama dengan OpenMP untuk dijalankan pada CPU multi-core. Beberapa metode di GAMESS telah diperbarui untuk menggunakan OpenMP secara opsional untuk

membongkar wilayah yang secara komputasi mengarah ke GPU. Penelitian Bak (2021) difokuskan pada port GPU metode HF dan RI-MP2 menggunakan OpenMP.

Adapun alur GAMESS adalah sebagai berikut. Metode HF memecahkan satu set persamaan nilai eigen non-linier iteratif untuk energi sistem molekuler. Ini memiliki dua langkah utama, yaitu: i) perhitungan sejumlah besar (pada urutan 4 yang merupakan ukuran sistem molekuler) dari 4-indeks 2-elektron integral (4-2ERI); dan ii) membentuk matriks 2 Fock dengan mengontraksi tensor 4 4-2ERI dengan matriks densitas. HF menggunakan metode dasar yang merupakan titik awal untuk banyak metode dengan akurasi yang lebih tinggi seperti metode RI-MP2. Dalam metode RI-MP2, 4-2ERI didekati sebagai produk dari 3-2ERI. Ini menyederhanakan integral evaluasi dari integral tolakan 2-elektron 4-indeks ke 3-indeks dan memungkinkan penggunaan operasi perkalian matriks yang efisien.

Selanjutnya, implementasi dan pengoptimalan dengan OpenMP adalah sebagai berikut. Untuk kode HF, penelitian Bak (2021) berfokus pada evaluasi 4-2ERI. Paralel dengan threading MPI+OpenMP pada CPU, kode mempertahankan beberapa tingkat pernyataan kondisional. tambahan, karena pekerjaan komputasi tidak ditugaskan secara merata ke utas, akan timbul kesulitan dari ketidakseimbangan beban. Untuk mengatasi ini, direorganisasi aliran kontrol dan urutan integral dihitung dengan menempatkan kondisional ke dalam blok kode terpisah dan menyortir integral sebelumnya. Hanya beberapa arahan OpenMP yang ditambahkan ke kode yang diatur ulang. Arahan OpenMP dimasukkan ke

kode offload untuk GPU (dan subrutin yang dipanggil dari wilayah target dianotasi dengan

'menyatakan target'). Diperhatikan bahwa rutinitas untuk menghitung integral, misalnya, int1, tidak dimodifikasi sama sekali. Versi GAMESS ini sedang dalam pengembangan cabang dan mendukung satu jenis integral. Untuk kode RI-MP2, difokuskan pada perhitungan koreksi perturbatif, yaitu didominasi oleh panggilan ke rutinitas perkalian matriks (DGEMM). Di sini, strategi untuk port dari CPU ke GPU adalah untuk menggabungkan bagian tertentu sehingga input ke panggilan DGEMM lebih besar, menghasilkan lebih tinggi intensitas aritmatika per panggilan DGEMM, dan peluncuran kernel yang lebih sedikit.

Studi Kasus 2

Bak (2021) juga membahas tentang GenASiS (*General Astrophysical Simulation System*). GenASiS adalah kode yang dikembangkan untuk simulasi skala besar fenomena astrofisika yang menargetkan superkomputer. Saat ini, ini GenASiS ditujukan untuk mensimulasikan dan memodelkan supernova core-collapse dan fenomena terkait peristiwa supernova yang dapat diamati. Simulasi menggunakan GenASiS telah menyebabkan penemuan amplifikasi medan magnet oleh Stasioner Accretion Shock Instability (SASI) di lingkungan supernova.

Implementasi GenASiS menggunakan OpenMP adalah sebagai berikut. Inti dari fasilitas penyimpanan data di GenASiS Basics adalah Kelas StorageForm, yang terdiri

dari anggota dan metode untuk menyimpan data generik dan metadata. Data disimpan sebagai dua dimensi array di mana indeks pertama biasanya menyebutkan sel-sel di mesh dan indeks kedua menyebutkan variabel. Metadata termasuk unit dan nama variabel yang dapat digunakan untuk I/O dan visualisasi. Sebagian besar pemecah dan kebutuhan penyimpanan GenASiS ditulis menggunakan kelas StorageForm, memungkinkan kode yang seragam dan disederhanakan untuk fungsionalitas seperti I/O dan pertukaran sel hantu tetangga terdekat.

Implementasi OpenMP dari GenASiS memperluas Kelas StorageForm dengan metode untuk mencerminkan, mengaitkan, dan menyinkronkan alokasi data memori host (CPU) dengan alokasi memori perangkat (GPU) yang sesuai. Gambar 1 mengilustrasikan penggunaan kelas StorageForm dengan metode yang relevan untuk dikelola alokasi dan asosiasi memori perangkat. Pada Gambar 1, baris 4 mengalokasikan anggota kelas array dua dimensi nilai pada memori host. Baris 5 mencerminkan alokasi tersebut pada perangkat dan mengaitkan lokasi memori di host dengan yang di perangkat. Ini dilakukan dengan menggunakan runtime perpustakaan OpenMP rutin `omp_target_associate_ptr()` di bawah tenda. Dengan asosiasi ini, runtime OpenMP bertemu dengan host itu variabel dalam wilayah target, pemetaan implisit, dan transfer data dihindari karena pemetaan tersebut sudah ada. Baris 7 menginisialisasi nilai anggota dengan kondisi khusus masalah. Baris 8 memperbarui salinan perangkat variabel dengan nilai awal, yang kemudian dapat digunakan di dalam rutin `AddKernel()` pada perangkat.

```

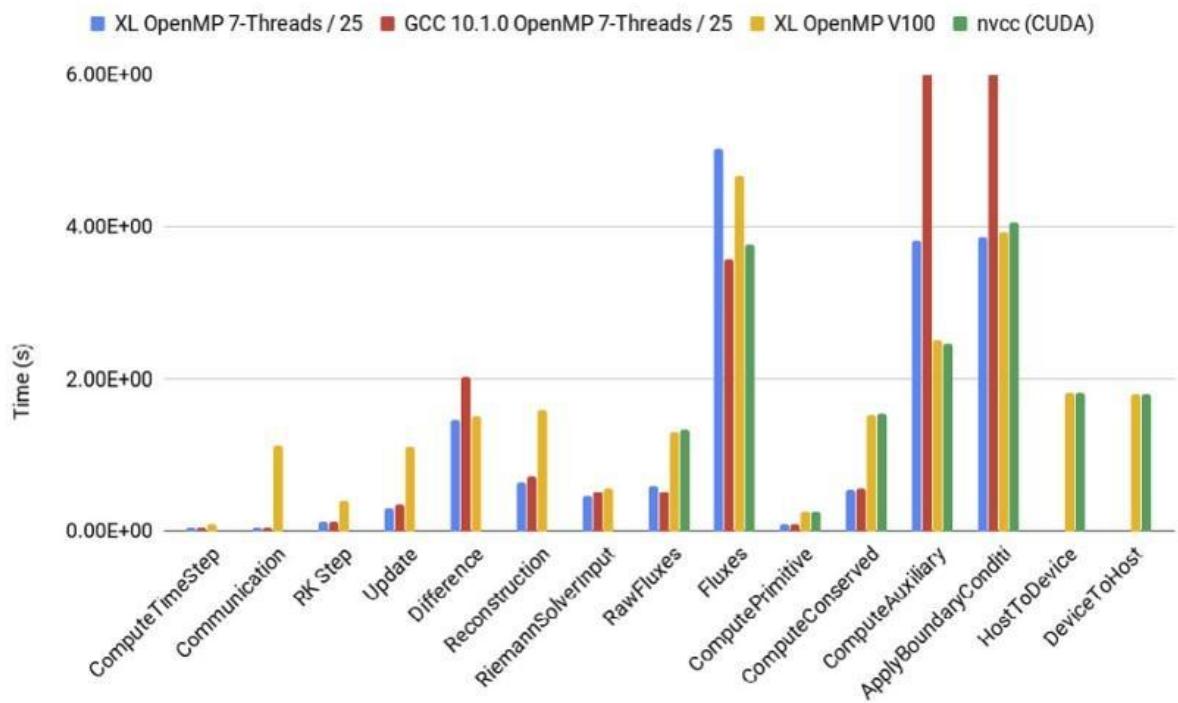
1 type ( StorageForm ) :: &
2     Fields
3 ...
4 call Fields % Initialize ( [ nCells, nVariables ], ... )
5 call Fields % AllocateDevice ( )
6 ...
7 call SetInitialConditions ( Fields % Value )
8 call Fields % UpdateDevice ( )
9 call AddKernel &
10    ( Fields % Value ( :, 1 ), &
11      Fields % Value ( :, 2 ), &
12      Fields % Value ( :, 3 ),
13      UseDevice = .true. )
14 call Fields % UpdateHost ( )

```

Gambar 1. Kode StorageForm untuk GenASiS

Berikut ini adalah pengaturan waktu untuk kernel komputasi dan transfer data di

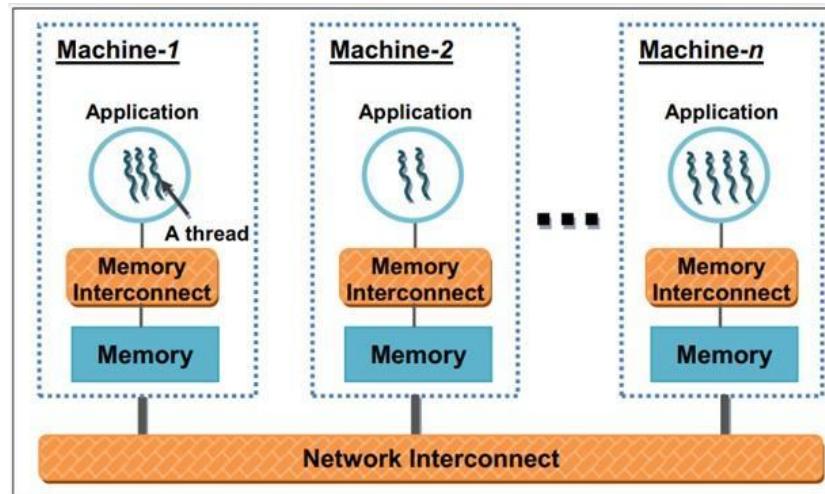
GenASiS Basics Riemann Problem 3D dengan 2563 sel untuk 50 siklus



2. Aplikasi *Distributed Memory*

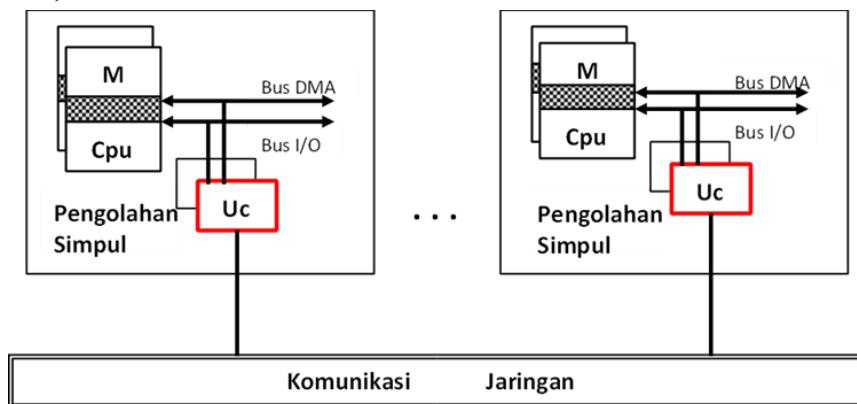
a. Definisi Distributed Memory

Sistem memori terdistribusi membutuhkan jaringan komunikasi untuk menghubungkan memori antar prosesor. Prosesor pada arsitektur memori ini mempunyai memori lokal sendiri dan alamat memori dalam satu processor yang tidak terhubung untuk processor lain, sehingga tidak ada konsep ruang alamat global di semua processor. Perubahan pada memori lokal tidak mempengaruhi memori lain, oleh karena itu konsep koherensi cache tidak berlaku. Jika diperlukan pemrosesan *interprocessor*, tugas *programmer* menentukan bagaimana dan kapan data akan dikomunikasikan. Sinkronisasi antara tugas-tugas adalah juga tanggung jawab programer. Jaringan “fabric” digunakan untuk transfer data yang bervariasi, meskipun dapat sesederhana seperti Ethernet. Dalam *distributed memory*, node atau simpul pemrosesan tidak dapat berbagi ruang memori secara fisik dan tidak dapat mengatasi memori node/simpul lainnya. *Distributed memory* memiliki I/O sebagai satu-satunya mekanisme sederhana untuk kerjasama node atau simpul dengan pertukaran nilai eksplisit dan memungkinkan memori bersama dapat ditiru.



Gambar 2.1 Ilustrasi Sistem *Distributed Memory*

Sumber: Aleem, 2015



Gambar 2.2 Ilustrasi Unit I/O yang didedikasikan untuk node interfacing (UC)

Sumber: <http://groups.di.unipi.it/~vannesch/SPA/SPA-13-Cluster.pdf>

Gambar di atas menunjukkan bahwa unit I/O yang didedikasikan untuk node atau simpul interfacing (UC) memiliki satuan komunikasi, satuan jaringan antarmuka, dan kartu jaringan. Distribusi memory memiliki beberapa jenis arsitektur memori, yaitu: PC/Workstation Cluster, Multicuster, Massively Parallel Processor (MPP), Grid, Data Center, Server Farm, Cloud.

Kelebihan menggunakan distributed memory adalah *scalable*, yaitu jumlah prosesor dan jumlah memori dapat dengan mudah ditingkatkan secara proporsional, setiap prosesor cepat dan dapat mengakses memorinya sendiri tanpa interferensi atau campur tangan dan tanpa overhead yang dikeluarkan dengan berusaha mempertahankan cache koherensi global; dan *cost effective*, yaitu dapat menggunakan komputer komoditas, processor off-the-rak dan jaringan.

Sedangkan Kelemahan menggunakan *distributed memory* adalah tugas *programmer* semakin sulit terkait detail komunikasi data yang dimana programer bertanggung jawab untuk banyak rincian yang berkaitan dengan komunikasi data antar processor; sulit untuk memetakan struktur data yang ada berdasarkan memori global, non-seragam akses memori sehingga data yang berlaku pada remote simpul membutuhkan waktu yang lebih lama untuk mengakses data lokal di simpul.

b. Aplikasi Distributed Memory

Ketersediaan kumpulan data besar telah memicu minat yang signifikan dalam menggunakan teknologi Big Data modern untuk memproses sejumlah besar informasi dalam klaster *distributed-memory* dari komoditas perangkat keras [Abuin, 2020]

1) Studi Kasus 1

Kumar [2022] melakukan penelitian berjudul *Memory-optimized distributed utility mining for big data*. Dengan berlatar belakang bahwa semakin menuju era modern, media sosial, layanan online smartphone, dan Internet of Things (IoT) menghasilkan data dalam jumlah besar (data terstruktur, tidak terstruktur, dan semi terstruktur) setiap detik dimana pendekatan tradisional tidak cukup untuk menangani jenis data tersebut secara efektif, sehingga Kumar mengusulkan pendekatan paralel bernama distributed memory optimized utility mining (DMOUM) untuk penambangan pola frekuensi tinggi berbasis utilitas untuk mengatasi persoalan big data.

Algoritma yang digunakan adalah struktur data berbasis array. Hasil kinerja DMOUM terhadap pemangkasan *high utility itemsets* (HUIs) akan dibandingkan dengan algoritma PHUI-Miner, mHUIMiner, dan DMOUM-Baseline. Langkah-langkah untuk yang dilakukan, yaitu membangun 1-HTWUIs, pemisahan database dan penugasan ke node pekerja, strategi pemangkasan, dan mengusulkan algoritma.

Algorithm 1: DMOUM(Distributed Memory Optimized Utility Mining)

Input: D:dataset, minThres: minimum utility threshold
Output: a collection of high-utility itemsets

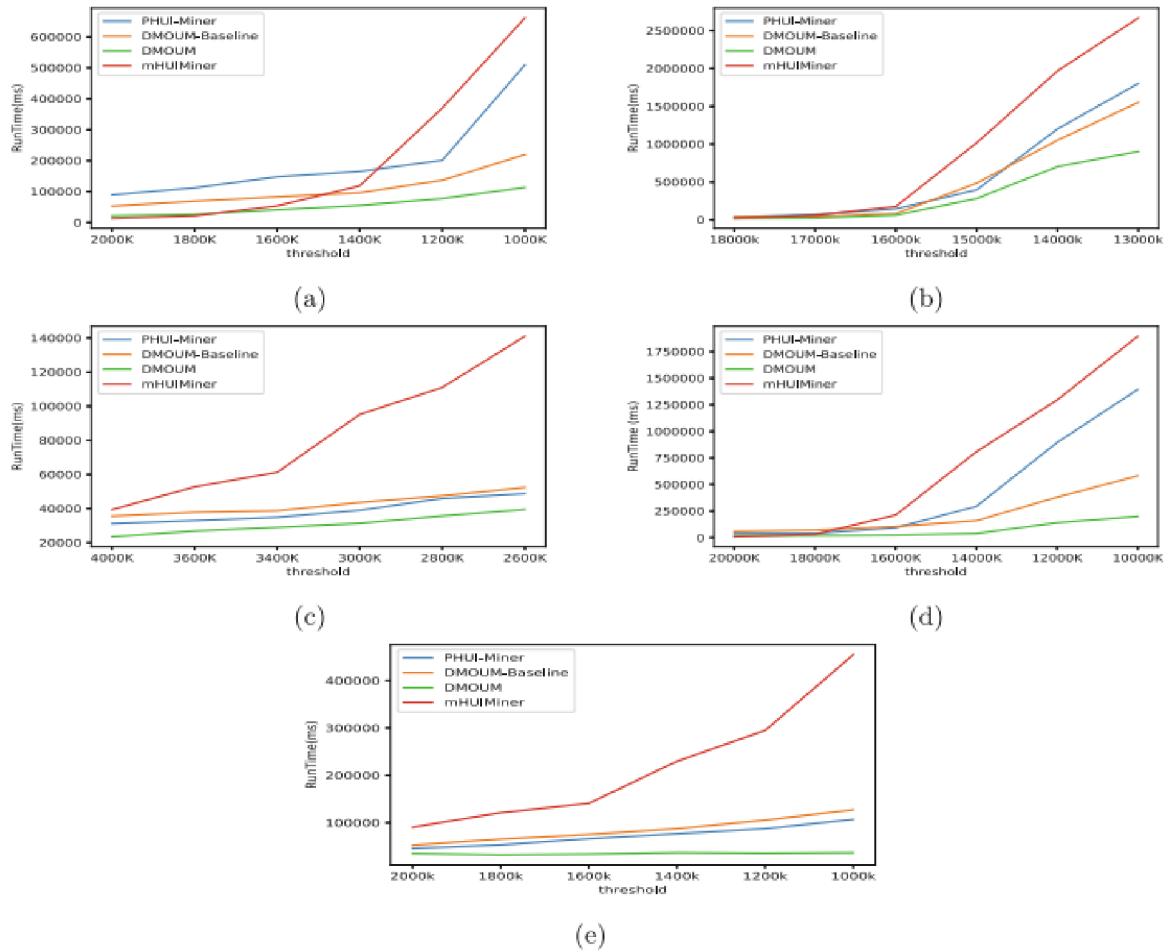
- 1: Compute the TWU for each item by scanning Database
- 2: Secondary(γ) = { $e|e \in E \wedge TWU(\gamma, e) \geqslant \text{minThres}$ }
- 3: Generate secondary list for itemset γ
- 4: **for all** transaction $t \in \text{Dataset}$ **do**
- 5: remove items \notin secondary list form each t
- 6: **if** $t.\text{count} == \text{NULL}$ **then**
- 7: remove t
- 8: **end if**
- 9: sort items in ascending order w.r.t TWU value in dataset
- 10: **end for**
- 11: Compute the sub-tree utility for items \in secondary list by scanning revised database
- 12: Generate primary-items for itemset γ using sub-tree utility
- 13: Filter primary and secondary items for each nodes
- 14: Generate Node-Data
- 15: **for all** $i \in$ number of nodes **do**
- 16: **for all** transactions $t \in \text{node-data}(i)$ **do**
- 17: **if** first item of $t \notin \text{secondary}(i)$ **then**
- 18: remove t from revised database
- 19: **end if**
- 20: **end for**
- 21: **end for**
- 22: **for all** $i \in$ number of nodes **do**
- 23: Local-miner(ϕ , node-data(i), primary(i), secondary(i),
 minThres)
- 24: **end for**

Ac

Go

Gambar 2.3 Algoritma DMOUM yang diusulkan penelitian [Kumar, 2022]

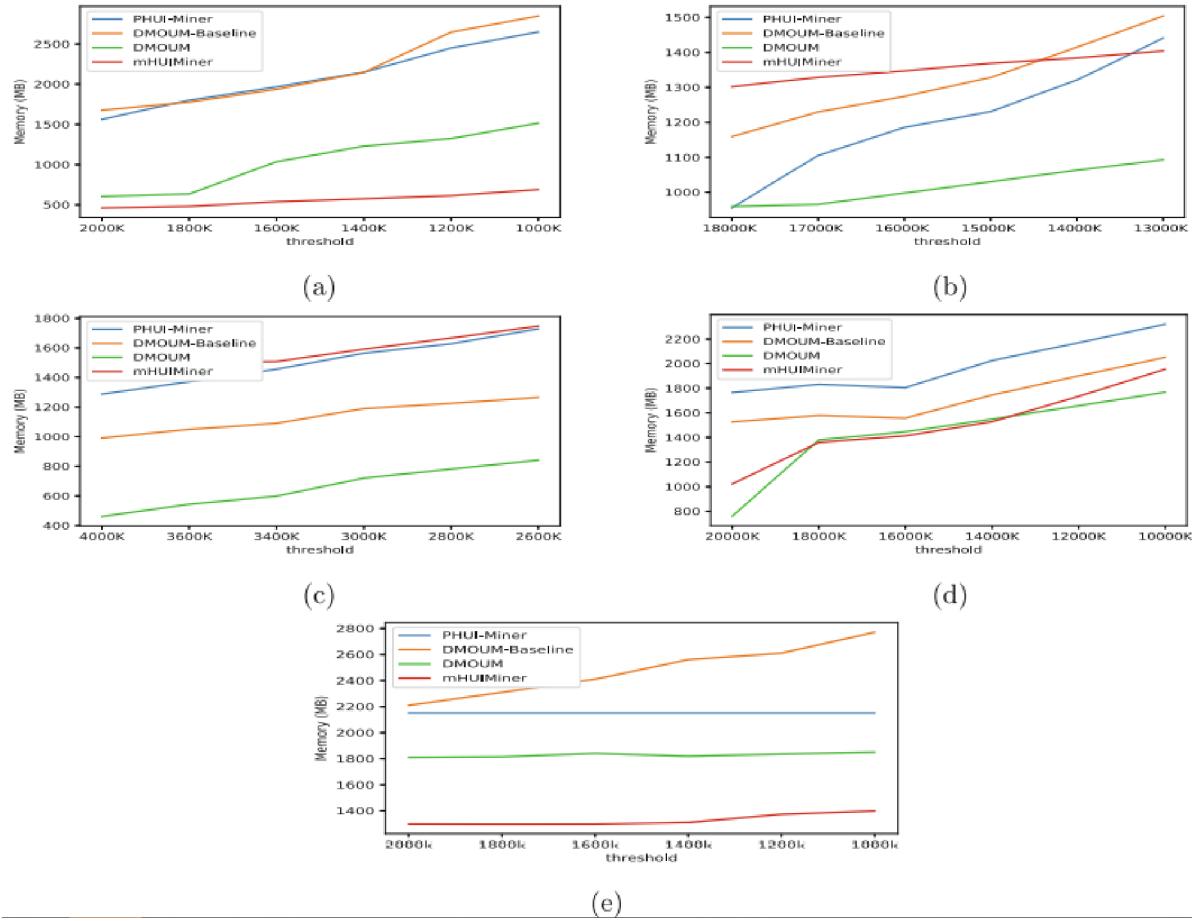
Hasil yang diperoleh dari penelitian tersebut adalah sebagai berikut.



Gambar 2.4 Hasil *running time* dari algoritma yang diusulkan dengan ambang utilitas minimum yang

bervariasi (a) Kosarak (b) Connect (c) Chainstore (d) Pumsb (e) T40110D100K

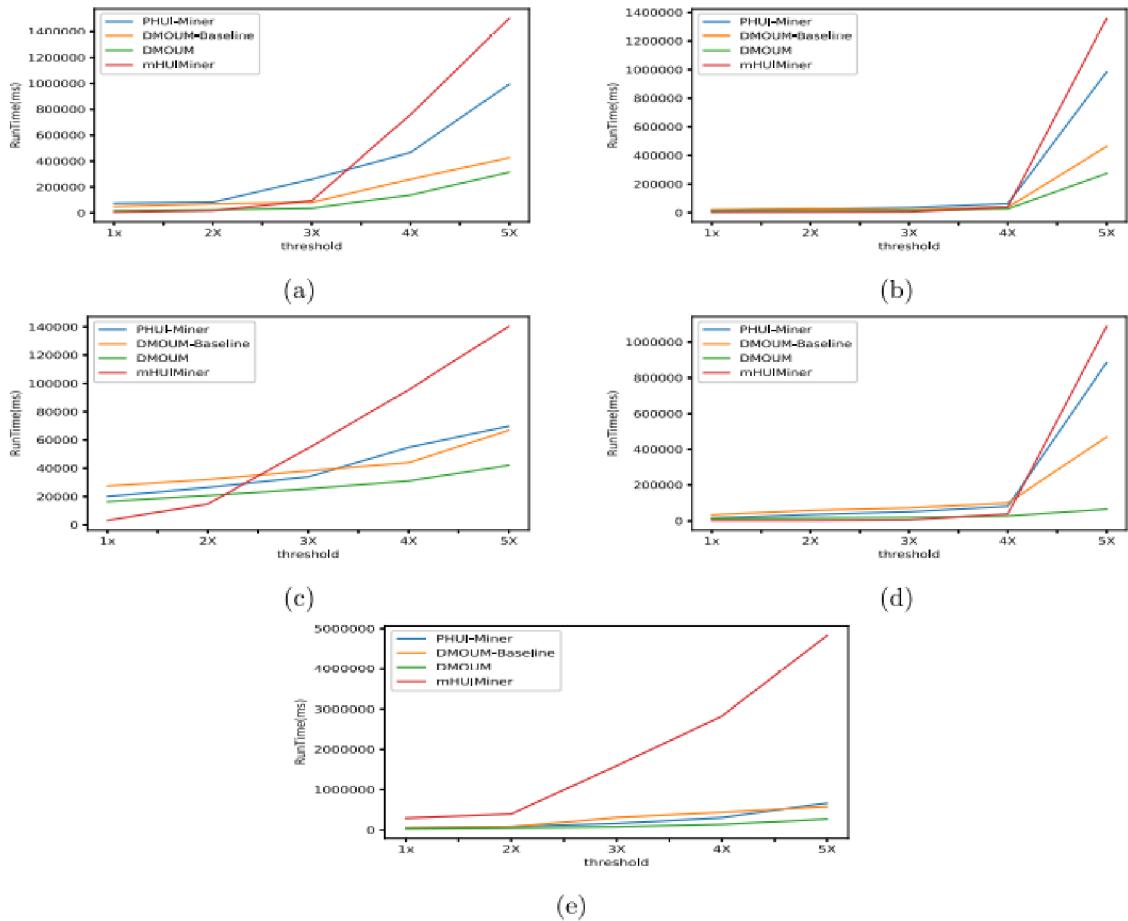
Gambar 2.4 menunjukkan bahwa algoritma yang diusulkan DMOUM secara signifikan mengungguli PHUI-Miner, dan algoritma mHUIMiner dan DMOUM-Baseline. Teknik pemangkasan oleh algoritma DMOUM secara signifikan mengurangi waktu yang dihabiskan pada pemangkasan HUIs dalam kumpulan data. Jumlah transaksi yang dikerjakan oleh DMOUM menghasilkan cakupan ekstraksi pola yang jauh lebih kecil karena pemangkasan. Namun, dalam beberapa kasus, *run-time* DMOUM jauh lebih lama daripada varian DMOUM-baseline dikarenakan proses pemangkasan membutuhkan lebih banyak memindai kumpulan data.



Gambar 2.5 Konsumsi penggunaan memori setiap algoritma yang dibandingkan dengan (a) Kosarak (b)

Connect (c) Chainstore (d) Pumsb (e) T40110D100K

Dari gambar 2.5, algoritma DMOUM mengkonsumsi memori lebih sedikit dibandingkan PHUI-Miner dan DMOUM-baseline. Namun, pada beberapa kasus DMOUM mengkonsumsi lebih banyak memori dibandingkan mHUI-Miner dikarenakan kerangka terdistribusi selalu membutuhkan memori lebih dibandingkan dengan algoritma sistem tunggal.



Gambar 2.6 Perbandingan skalabilitas setiap algoritma pada dataset (a) Kosarak (b) Connect (c) Chainstore
 (d) Pumsb (e) T4110D100K

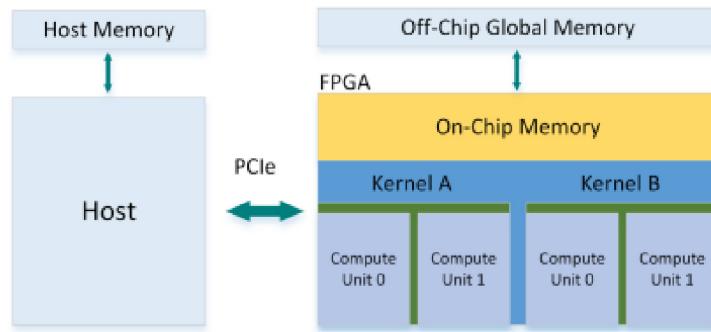
Keempat algoritma dieksekusi dengan ukuran data yang berbeda untuk memperkirakan skalabilitas ritme algoritma. Gambar 2.6 menunjukkan bahwa *run time* meningkat secara linear dengan peningkatan ukuran data. Hal itu disebabkan karena ada lebih banyak entri dalam daftar kandidat dan ada kebutuhan untuk lebih banyak waktu eksekusi sebagai akibat dari peningkatan jumlah transaksi. Waktu eksekusi yang meningkat secara eksponensial 4x dan 5 kali mengartikan kandidat yang meningkat secara agresif menghasilkan tugas yang jauh lebih banyak per *node worker*, yang meningkatkan waktu tunggu pekerjaan.

Kesimpulan dari penelitiannya yang mengusulkan algoritma paralel baru, yaitu DMOUM, untuk penambangan terdistribusi dari pola utilitas tinggi untuk big data adalah membuktikan bahwa pendekatan yang diusulkan memiliki kinerja yang lebih baik dalam hal *run-time*, konsumsi memori, dan ukuran skalabilitas, sehingga dapat memberikan solusi untuk masalah *real time* seperti kesehatan, pendidikan, e-commerce, dan sebagainya.

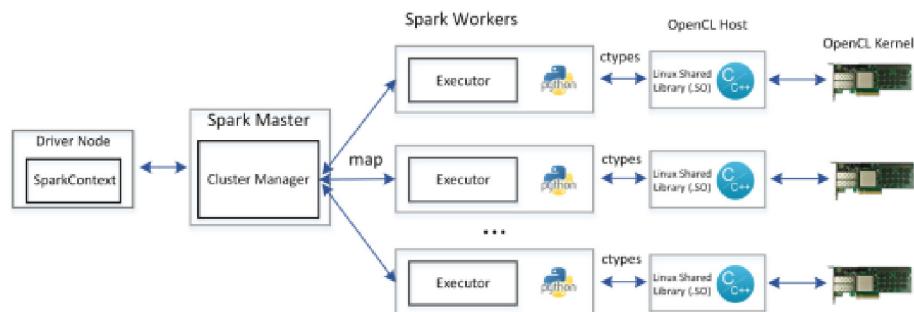
2) Studi Kasus 2

Junjie Hou [2018] melakukan penelitian berjudul *Design and implementation of reconfigurable acceleration for in-memory distributed big data computing*. Dengan berlatar belakang bahwa Spark (kerangka kerja komputasi terdistribusi yang efisien

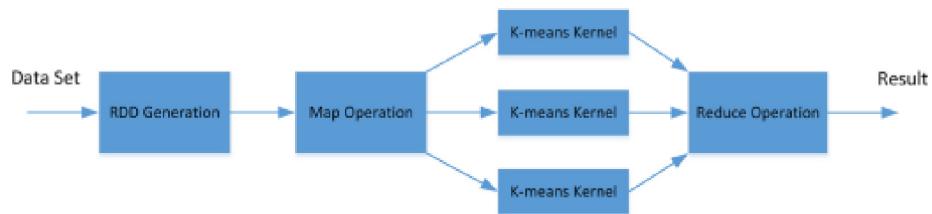
untuk pemrosesan data besar) hanya dilakukan pada CPU. Karena tingkat paralelisme yang rendah dan inefisiensi daya CPU dapat membatasi kinerja dan skalabilitas klaster, maka Junjie dkk mengusulkan kerangka kerja untuk mengusulkan metode untuk menghubungkan Spark dengan aplikasi OpenCL yang merupakan standar untuk lintas platform, pemrograman paralel dari beragam prosesor dan banyak digunakan dalam komputasi heterogen, dan menggunakan FPGA untuk mempercepat tugas Spark yang dikembangkan dengan Python. Kinerja dan energi efisiensi framework Spark berbasis FPGA digambarkan pada studi kasus percepatan algoritma K-means.



Gambar 2.6 Akselerator FPGA dalam model OpenCL



Gambar 2.7 Metodologi keseluruhan dari integrasi FPGA ke kerangka kerja Spark



Gambar 2.8 Proses mengoptimalkan *K-means clustering* Melalui penelitian tersebut, diperoleh hasil berikut:

- a) Waktu eksekusi K-means pada CPU dan Spark berbasis FPGA

	CPU based Spark	FPGA based Spark	Speedup
Performance	252 s	72 s	3.5

- b) Efisiensi energi dari *slave node* pada CPU dan Spark berbasis FPGA

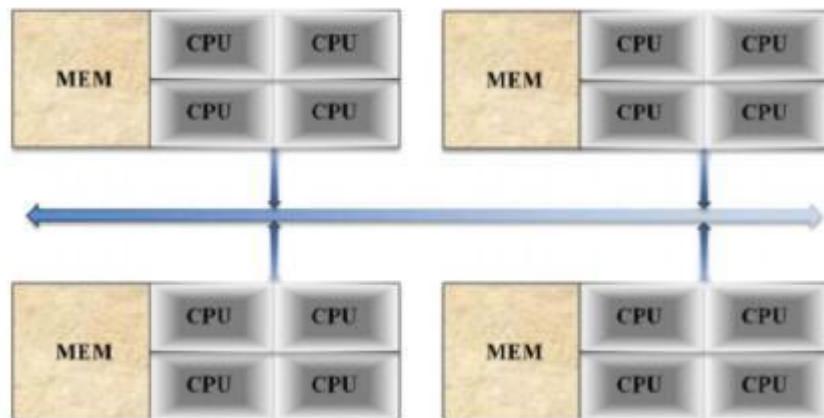
Energy efficiency (4.06x)	CPU based Spark	FPGA based Spark
Power (W)	177.86	153.34
Power dissipation (J)	177.86 * 3.5t	153.34 * 1t

Hasilnya menunjukkan bahwa implementasi Spark berbasis FPGA dari K-means mencapai kecepatan 3,5 kali dan efisiensi energi 4,06 kali dibandingkan kerangka kerja Spark asli.

3. Aplikasi *Hybrid Memory*

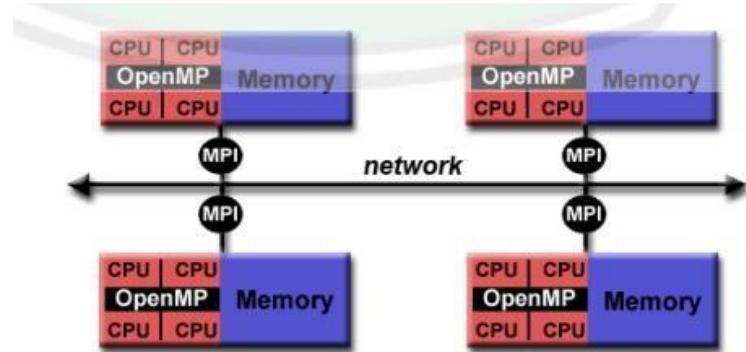
a. Definisi Hybrid Memory

Model Hybrid menggabungkan dua model pemrograman dari shared memory dan distributed memory. Setiap node unit adalah sistem memori bersama ada dua atau lebih multiprosesor simetris (SMP) berbagi hal yang sama sumber daya memori. Kemudian, sistem distributif dibentuk yang menghubungkan node memori bersama melalui jaringan (Chew, 2013).



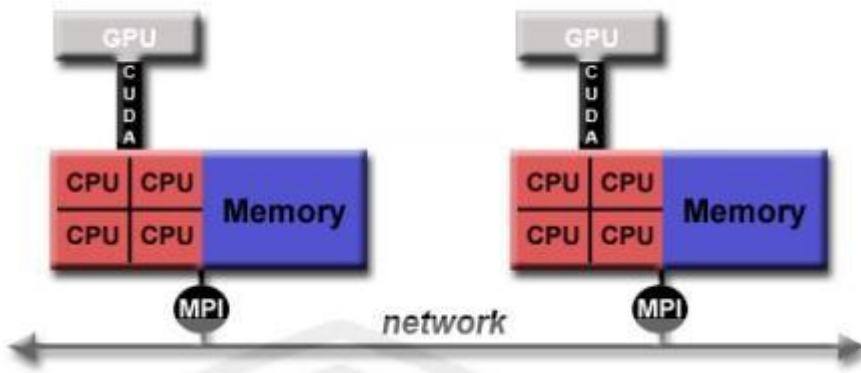
Gambar: Arsitektur hybrid distributed-shared memory.

Pada saat ini, Kebanyakan superkomputer menggunakan sistem hybrid. contoh umum dari model hybrid adalah kombinasi dari message passing model (MPI) dengan model thread (OpenMP). Model hybrid ini cocok dengan arsitektur perangkat keras berkelas multi-core yang paling populer (saat ini). Contoh lain yang serupa dan semakin populer dari model hibrida adalah menggunakan MPI dengan pemrograman CPU-GPU (Graphics Processing Unit). Pertukaran data antara memori node-local dan GPU menggunakan CUDA (atau sesuatu yang setara) (Ardinati dkk, 2014)



Gambar: Model pemrograman Hybrid kombinasi message passing (MPI) dan thread (OpenMP)

Sumber: Ardianti dkk, 2014



Gambar: Model pemrograman Hybrid kombinasi message passing (MPI) dan CPU-GPU(CUDA)

Sumber: Ardianti dkk, 2014

b. Aplikasi (Studi Kasus) parallel Computing berbasis Hybrid Memory

Studi Kasus 1

Cinderatama dkk, 2018 melakukan penelitian dengan judul desain dan implementasi hybrid cloud computing sebagai infrastruktur untuk analisis big data menggunakan analytic hierarchy process (AHP). Penelitian ini dilakukan karena dalam implementasi big data dibutuhkan sumber daya yang cukup besar untuk dapat melakukan analisis terhadap data-data yang jumlahnya sangat besar tersebut, hal ini biasanya menjadi kendala dikarenakan keterbatasan sumber daya yang dimiliki. Komputasi awan (cloud computing) yang salah satunya mempunyai sifat elasticity di dalamnya, menawarkan solusi keterbatasan sumber daya ini. Sumber daya yang terbatas misalkan dalam hal processor, RAM atau storage, dapat digabungkan dengan sumber daya yang dimiliki public cloud provider yang tersedia di market. Sehingga penggabungan 2 sumber daya ini, private cloud dan public cloud, dimana

penggabungan ini dapat menjadi solusi untuk dapat mengimplementasikan analisis big data yang dapat diterapkan untuk analisis berbagai macam bidang. Secara khusus tujuan penelitian ini adalah merumuskan sebuah metode minimalisasi cost dalam pemilihan public cloud dengan pendekatan sistem pendukung keputusan menggunakan metode AHP pada pemilihan public cloud.

Langkah pertama yang dilakukan dalam penelitian ini adalah pengumpulan data cost penggunaan resource dari public cloud. Selanjutnya dilakukan analisis kebutuhan sumber daya yang diperlukan untuk melakukan analisis big data dengan studi kasus topik tertentu. Selanjutnya tahap analisis terhadap pemilihan public cloud yang tepat untuk digunakan sumber dayanya dengan pertimbangan minimalisasi cost. Langkah terakhir adalah implementasi hybrid cloud dan melakukan analisis dan evaluasi terhadap metode yang diusulkan.

Penelitian Cinderatama dkk, 2018 ini, data pengujian yang digunakan adalah data dari public cloud provider yang diambil dari beberapa top cloud provider menurut survey yang dilakukan oleh website channele2e.com dan clutch.co(<https://clutch.co/cloud> dan [https://www.channele2e.com/channel-partners/program/top-10-iaas-clouds-csp-reviews-rankings -amazon-azure-google-ibm/](https://www(channele2e.com/channel-partners/program/top-10-iaas-clouds-csp-reviews-rankings-amazon-azure-google-ibm/)). Data cloud provider yang akan digunakan adalah data yang diambil pada bulan November 2017.

Dimana jumlah parameter masukan(input) yang digunakan sebanyak 5 kandidat, yaitu processing unit(CPUcore dan CPU speed), memory unit, storage unit, dan price. Dalam penelitian ini, akan dilakukan perhitungan dari data yang sudah diambil untuk mendapatkan hasil rekomendasi public cloud provider terbaik untuk selanjutnya digabungkan dengan private cloud guna mewujudkan sebuah infrastruktur hybrid cloud.

id	Cp	paket	proc_core	proc_speed	ram	storage	price	bandwidth	datacenter
1	1and1	Cloud server M	1	2.00	1.00	50	9.99	300 mbps up and down	USA, Germany, Spain
2	1and1	Cloud server L	2	2.00	1.00	80	19.99	300 mbps up and down	USA, Germany, Spain
...
13	Digital ocean	1	1	2.00	0.51	20	5.00	1TB	New York, San Francisco, Amsterdam, Singapore, London
14	Digital ocean	2	1	2.00	1.00	30	10.00	2TB	New York, San Francisco, Amsterdam, Singapore, London
...
15	Digital ocean	3	2	2.00	2.00	40	2.00	3TB	New York, San Francisco, Amsterdam, Singapore, London
...
53	IBM Cloud – Softlayer	B1.16x 64x100	16	2.00	64.00	100	588.65	250Mbps	US

Tabel. Data Public Cloud Provider

Selanjutnya dari data dilakukan Analytic Hierarchy Process (AHP) dengan membuat matriks tabel penentuan bobot, membuat matriks tabel pair-wire comparison dan membuat matriks tabel overall composite weight. Dalam langkah-langkah penting di atas kemudian akan didapatkan pula beberapa nilai yaitu Priority Vector, Principal Eigenvalue (Imax), Consistency Index (CI), Consistency Ratio (CR) dimana Jika nilainya kurang dari 0,1 atau 10%, maka bobot yang kita berikan sudah konsisten. Dengan hasil yang diperoleh untuk Analytic Hierarchy Process (AHP) Pair Wise Comparison adalah sebuah composite weights seperti pada tabel dibawah, yang digunakan untuk menentukan bobot dari masing-masing public cloud provider yang sudah ada dalam database, nilai ini nantinya digunakan untuk pemeringkatan prioritas public cloud provider yang disarankan untuk digunakan berdasarkan metode AHP yang diimplementasikan.

Overall Composite Weight	Weight	CP1	CP2	CPx
Processing_Core	0.2	0.002906976744 1860465	0.0019346850 332765816	...	0.005813953488 372093
Processing_Speed	0.2	0.005813953488 372093	0.0038693700 66553163	...	0.006653174073 7409435
RAM	0.2
Storage	0.2
Price	0.2	0.011627906976 744186	0.0145101377 49574381	...	0.011941158365 1279
Composite Weight		0.0240290747 6818176	0.016915942 17271036	...	0.0291396086 44773437

Tabel. Overall Composite Weight

Untuk melihat hasil dari analisis AHP, analytic hierarchy di implementasikan dalam bahasa java dengan source code yang digunakan dalam pengujian sistem pendukung keputusan AHP, yaitu:

```

private void concludeFinalValues() {
    ArrayList<Double> compositeWeights = DoubleMatrix.arrayOfZeroes(this.alternatives.size());

    int i;
    for(i = 0; i < this.overallCompositeWeight.getRowCount(); ++i) {
        double weight = ((Double)this.overallCompositeWeight.get(i, 0)).doubleValue();

        for(int col = 1; col < this.overallCompositeWeight.getColumnCount(); ++col) {
            double cell = ((Double)this.overallCompositeWeight.get(i, col)).doubleValue();
            double cw = ((Double)compositeWeights.get(col - 1)).doubleValue() + weight * cell;
            compositeWeights.set(col - 1, cw);
        }
    }
    System.out.println("Composite Weights");

    for(i = 0; i < compositeWeights.size(); ++i) {
        Double d = (Double)compositeWeights.get(i);
        System.out.print(d + ", ");
        ((Alternative)this.alternatives.get(i)).setFinalValue(d.doubleValue());
    }
    System.out.println("\n");
}
private void populateComparisonMatrices() {
    Iterator var1 = this.criteriaList.iterator();

    while(var1.hasNext()) {
        Criteria c = (Criteria)var1.next();
        ArrayList<ScoredLabel> scoredLabels = Alternative.allScoredLabelsForCriteria(c, this.alternatives);
        PairWiseMatrix matrix = new PairWiseMatrix(scoredLabels);
        matrix.autoFillValues();
        System.out.println(c.getName().toUpperCase());
        System.out.println(matrix.toString());
        this.comparisonMatrices.add(matrix);
    }
}
public ArrayList<Alternative> getSortedAlternatives(final boolean descending) {
    this.alternatives.sort(new Comparator<Alternative>() {
        public int compare(Alternative a1, Alternative a2) {
            return !descending ? Double.compare(a1.getFinalValue(), a2.getFinalValue()) : Double.compare(a2.getFinalValue(),
a1.getFinalValue());
        }
    });
    return this.alternatives;
}
}

```

Gambar. Source Code Perhitungan final dan hasil AHP

Dengan hasil analisis dari program AHP yang telah diimplementasikan menggunakan java, yaitu:

```

PROCESSINGCORE
1.0, 0.5, 0.5, 0.25, 0.125, 0.0833333333333333, 0.0625, 0.5, 0.25, 0.125, 0.0625,
0.03125, 1.0, 1.0, 0.5, 0.5, 0.25, 0.125, 0.0833333333333333, 0.0625, 0.05, 1.0, 0.5,
0.25, 0.125, 0.0625, 0.03125, 0.015625, 1.0, 1.0, 0.5, 0.25, 0.125, 1.0, 1.0, 1.0, 0.5,
0.5, 0.5, 0.5, 0.25, 0.25, 0.25, 0.125, 0.125, 0.125, 0.125, 0.0625, 0.0625, 0.0625,
0.0625, 0.0208333333333332, 0.0208333333333332
2.0, 1.0, 1.0, 0.5, 0.25, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.0625,
2.0, 1.0, 1.0, 0.5, 0.25, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.0625,
0.0625, 0.03125, 2.0, 2.0, 1.0, 0.5, 0.25, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.5, 0.5,
0.5, 0.5, 0.25, 0.25, 0.25, 0.125, 0.125, 0.125, 0.04166666666666664,
0.04166666666666664
2.0, 1.0, 1.0, 0.5, 0.25, 0.1666666666666666, 0.125, 1.0, 0.5, 0.25, 0.125, 0.0625, 2.0,
2.0, 1.0, 0.5, 0.25, 0.1666666666666666, 0.125, 0.1, 2.0, 1.0, 0.5, 0.25, 0.125,
0.0625, 0.03125, 2.0, 2.0, 1.0, 0.5, 0.25, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.5, 0.5,
0.5, 0.5, 0.25, 0.25, 0.25, 0.125, 0.125, 0.125, 0.04166666666666664,
0.04166666666666664
4.0, 2.0, 2.0, 1.0, 0.5, 0.3333333333333333, 0.25, 2.0, 1.0, 0.5, 0.25, 0.125, 4.0, 4.0,
2.0, 1.0, 0.5, 0.3333333333333333, 0.25, 0.2, 4.0, 2.0, 1.0, 0.5, 0.25, 0.125,
0.0625, 4.0, 4.0, 2.0, 1.0, 0.5, 4.0, 4.0, 4.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 0.5,
0.5, 0.5, 0.25, 0.25, 0.25, 0.0833333333333333, 0.0833333333333333
8.0, 4.0, 4.0, 2.0, 1.0, 0.6666666666666666, 0.5, 4.0, 2.0, 1.0, 0.5, 0.25, 8.0, 8.0, 4.0,
4.0, 2.0, 1.0, 0.6666666666666666, 0.5, 0.4, 8.0, 4.0, 2.0, 1.0, 0.5, 0.25, 0.125, 8.0, 8.0,
4.0, 2.0, 1.0, 8.0, 8.0, 8.0, 4.0, 4.0, 4.0, 2.0, 2.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 0.5,
0.5, 0.5, 0.1666666666666666, 0.1666666666666666
12.0, 6.0, 6.0, 3.0, 1.5, 0.75, 8.0, 3.0, 1.5, 0.75, 3.0, 12.0, 12.0, 6.0, 6.0, 3.0, 1.5,
1.0, 0.75, 0.6, 12.0, 6.0, 3.0, 1.5, 0.75, 0.375, 0.1875, 12.0, 12.0, 6.0, 3.0, 1.5, 12.0,
12.0, 12.0, 6.0, 6.0, 3.0, 3.0, 3.0, 3.0, 1.5, 1.5, 1.5, 1.5, 0.75, 0.75, 0.75,
0.75, 0.25, 0.25
16.0, 8.0, 8.0, 2.0, 1.0, 3333333333333333, 1.0, 8.0, 4.0, 2.0, 1.0, 0.5, 16.0, 16.0, 8.0,
8.0, 4.0, 2.0, 1.3333333333333333, 1.0, 0.8, 16.0, 8.0, 4.0, 2.0, 1.0, 0.5, 0.25, 16.0,
16.0, 8.0, 4.0, 2.0, 16.0, 16.0, 16.0, 8.0, 8.0, 4.0, 4.0, 4.0, 2.0, 2.0,
2.0, 2.0, 1.0, 1.0, 0.5, 0.3333333333333333, 0.3333333333333333

```

Gambar. Matriks Parameter Processing Core

Hasil dari matrix parameter yang lain yaitu processing speed, memory, storage dan price, seperti berikut:

```

PRICE
1.0, 2.0010010010008, 3.0020020020020017, 5.0040040040040035,
13.012012012013, 24.024924924924927, 35.03403403403403,
2.8028028028028027, 5.805605605605605, 11.21121121121121,
22.42242242242242, 44.84484484484484, 0.5005005005005004,
1.0010010010009, 2.0020020020020017, 4.0040040040040035,
8.00600000800007, 16.016016016016014, 32.03203203203203,
48.048048048045, 64.06406406406406, 2.429429429429429,
4.859859859859859, 9.70970970970971, 19.41941941941942, 38.83883883883884,
77.67767767767768, 155.45545545545545, 1.3413413413413413,
1.7137137137135, 5.883883883883883, 13.108108108108107,
26.215215215215213, 3.5205205205205203, 3.5205205205205203,
5.5135135135135135, 5.513513513513513, 6.974974974974975, 6.974974974974975,
7.439439439439438, 7.439439439439438, 13.35235235235235, 13.35235235235235,
15.013013013013012, 15.013013013013012, 26.041041041041037,
26.041041041041037, 32.086086086086084, 32.086086086086084,
47.69669669696967, 47.696696969696967, 58.923923923923915,
146.94394394394394, 146.94394394394394,
0.4997498749374689, 1.0, 1.5002501250625313, 2.500750375187594,
6.502751375687845, 12.45622811405703, 17.508254127063534,
1.400700301750876, 2.801400700301753, 5.602801400700351,
11.205602801400701, 22.411205602801402, 0.25012506253128585,
0.5002501250625313, 1.0005002501250626, 2.001000500250125,
4.00200100050025, 8.0040020010005, 16.008004002001, 24.012006003001503,
32.016008004002, 1.2141070535287635, 2.428714357178589, 4.852426213106554,
9.704852426213108, 19.409704852426216, 38.81940970405243, |

```

Gambar. Matriks Parameter Price

Dari penyusunan matrix pada masing-masing parameter tersebut akan dihasilkan sebuah composite weights yang digunakan untuk menentukan bobot dari masing-masing public cloud provider yang sudah ada dalam database, nilai ini nantinya digunakan untuk pemeringkatan prioritas public cloud provider yang disarankan untuk digunakan berdasarkan metode AHP yang diimplementasikan.

Composite Weights

0.02402907476818176, 0.01691594217271036, 0.015675522799064916,
0.016577679780148825, 0.020932536904668304, 0.029701000454057918,
0.039583058184953564, 0.014916291352855264, 0.016123972137759384,
0.02310015211332731, 0.039429655519006845, 0.0406658458905785,
0.02334464099661628, 0.015578169285151428, 0.012194438509991884,
0.01257910266340284, 0.01800976513991923, 0.02820950911578374,
0.038779030962355056, 0.04944099727660305, 0.013440624567445315,
0.011090132143224327, 0.011877876934338102, 0.016190571560327275,
0.02618833229121743, 0.04687003949245365, 0.019032443119182896,
0.017014407667690395, 0.011253057657632218, 0.013428794792894017,
0.02043096727878588, 0.010668831736515257, 0.013168831736515256,
0.009231507150205117, 0.011731507150205118, 0.009137701716589688,
0.011637701716589687, 0.009752543081179904, 0.012252543081179902,
0.00985773692057693, 0.012357736920576928, 0.011258297111642307,
0.013758297111642306, 0.013082704746861927, 0.015582704746861926,
0.016049659963705817, 0.018549659963705816, 0.020519592161785557,
0.02301959216178556, 0.026639608644773435, 0.029139608644773437,

Gambar. Composite Weights

Berdasarkan composite weights diatas hasil akhir dari analisis AHP yang sudah diimplementasikan menggunakan java ini adalah rekomendasi pemilihan public cloud yang dapat dipilih untuk implementasi hybrid cloud. Dari hasil analisis, diperoleh sumber daya public cloud yang terpilih adalah public cloud Digital Ocean, berdasarkan rekomendasi ini maka untuk tahap pengujian selanjutnya akan dibangun hybrid cloud dengan menggabungkan sumber daya dari private dan sumber daya dari public cloud Digital Ocean.

```

Digital Ocean (9) -> 0.04944099727880305
Google Cloud (n1-standard-32) -> 0.04687003940245385
Digital Ocean (1) -> 0.0406658458905785
1and1 (Cloud Server 5XL) -> 0.039583058184953564
OVH (B2-60) -> 0.039429655519006845
Digital Ocean (8) -> 0.038779030962355056
1and1 (Cloud Server 4XL) -> 0.029701000454057918
IBM Cloud - Softlayer (B1.16x84x100) -> 0.029139808844773437
Digital Ocean (7) -> 0.02820950911578374
IBM Cloud - Softlayer (B1.16x64x25) -> 0.026639608844773435
Google Cloud (n1-standard-16) -> 0.02618833229121743
1and1 (Cloud Server M) -> 0.02402907476818176
Digital Ocean (2) -> 0.02334464099861628
OVH (B2-30) -> 0.02310015211332731
IBM Cloud - Softlayer (B1.16x32x100) -> 0.02301959216178556
1and1 (Cloud Server 3XL) -> 0.020932536904666304
IBM Cloud - Softlayer (B1.16x32x25) -> 0.020519592161785557
Azure (A4) -> 0.02043098727878588
Azure (A0) -> 0.019032443119182896
IBM Cloud - Softlayer (B1.8x32x100) -> 0.018549659963705816
Digital Ocean (8) -> 0.01800978513091923
Azure (A1) -> 0.017014407667890395
1and1 (Cloud Server L) -> 0.01691594217271036
1and1 (Cloud Server XXL) -> 0.016577679780148825
Google Cloud (n1-standard-8) -> 0.016190571560327275
OVH (B2-15) -> 0.016123972137759384
IBM Cloud - Softlayer (B1.8x32x25) -> 0.016049659963705817
1and1 (Cloud Server XL) -> 0.015675522799064916
IBM Cloud - Softlayer (B1.8x16x100) -> 0.015682704746881926
Digital Ocean (3) -> 0.015578189285151428
OVH (B2-7) -> 0.014916291352855264
IBM Cloud - Softlayer (B1.4x16x25) -> 0.013768297111642306
Google Cloud (n1-standard-1) -> 0.013440824567445315
Azure (A3) -> 0.013428794792894017
IBM Cloud - Softlayer (B1.1x2x100) -> 0.013168831738515256
IBM Cloud - Softlayer (B1.8x16x25) -> 0.013082704746881927
Digital Ocean (5) -> 0.01257910266340284
IBM Cloud - Softlayer (B1.4x8x100) -> 0.012357738920578928
IBM Cloud - Softlayer (B1.2x8x100) -> 0.012252543081179902
Digital Ocean (4) -> 0.012194438509991884
Google Cloud (n1-standard-4) -> 0.011877878934338102
IBM Cloud - Softlayer (B1.1x4x100) -> 0.011731507150205118
IBM Cloud - Softlayer (B1.2x4x100) -> 0.011637701716589687
IBM Cloud - Softlayer (B1.4x16x25) -> 0.011268297111642307
Azure (A2) -> 0.011253057857832218

```

Gambar. Rekomendasi Public Cloud

Dari hasil yang diperoleh dapat diambil beberapa kesimpulan yakni sebagai berikut:

1. Telah dilakukan uji coba implementasi Analytic Hierarchy Process (AHP) menggunakan bahasa pemrograman Java. Pada percobaan tersebut rekomendasi public cloud terpilih yang dapat diimplementasikan dalam infrastruktur hybrid cloud.
2. Aplikasi Desktop AHP telah dibangun menggunakan perangkat lunak xampp, mysql, IntelliJ IDEA.
3. Implementasi Hybrid Cloud memanfaatkan teknologi Docker yang diimplementasikan dalam container-container.

Studi Kasus 2

Lai dkk, 2020 dalam penelitiannya Hybrid MPI and CUDA Parallelization for CFD Applications on Multi-GPU HPC Clusters memodelkan pemrograman hybrid kombinasi message passing (MPI) dan CPU-GPU (CUDA) dimana Unit pemrosesan grafis (GPU)

memiliki kemampuan floating-point yang kuat dan bandwidth memori yang tinggi dalam paralelisme data dan telah banyak digunakan dalam komputasi kinerja tinggi (HPC). Compute unified device architecture (CUDA) digunakan sebagai paralel platform komputasi dan model pemrograman untuk GPU untuk mengurangi kerumitan pemrograman. GPU dapat diprogram menjadi populer dalam aplikasi dinamika fluida komputasi (CFD). Penelitian ini mengusulkan algoritma paralel hybrid antarmuka penyampaian pesan dan CUDA untuk aplikasi CFD pada cluster HPC multi-GPU. skema AUSM + UP upwind dan metode Runge–Kutta tiga langkah masing-masing digunakan untuk diskritisasi spasial dan diskritisasi waktu. Bergolak solusi diselesaikan dengan model dua persamaan K SST. CPU hanya mengelola eksekusi GPU dan komunikasi, dan GPU bertanggung jawab untuk pemrosesan data. Eksekusi paralel dan optimasi akses memori digunakan untuk mengoptimalkan Kode CFD berbasis GPU. penelitian ini mengusulkan metode komunikasi tanpa pemblokiran untuk sepenuhnya tumpang tindih dengan komputasi GPU, CPU_CPU komunikasi, dan transfer data CPU_GPU dengan membuat dua aliran CUDA. Selanjutnya, domain satu dimensi metode dekomposisi digunakan untuk menyeimbangkan beban kerja antar GPU. Akhirnya, kami mengevaluasi algoritma paralel hibrida dengan aliran turbulen kompresibel di atas pelat datar. kinerja implementasi GPU tunggal dan skalabilitas multi-GPU cluster dibahas. Pengukuran kinerja menunjukkan bahwa paralelisasi multi-GPU dapat mencapai kecepatan lebih dari 36 kali sehubungan dengan komputasi paralel berbasis CPU, dan algoritma paralel memiliki skalabilitas yang baik.

MPI dan OpenMP adalah dua pemrograman aplikasi antarmuka yang banyak digunakan untuk menjalankan kode paralel pada platform multi-GPU. OpenMP dapat digunakan dalam node untuk paralelisasi berbutir halus menggunakan memori bersama, dan MPI bekerja pada memori yang dibagikan dan didistribusikan dan banyak digunakan untuk komputasi paralel besar-besaran. Secara umum, MPI menunjukkan penurunan kinerja dibandingkan dengan OpenMP tetapi relatif mudah dilakukan pada berbagai jenis perangkat keras dan memiliki skalabilitas yang baik. Penelitian ini, berbasis MPI komunikasi untuk memori bersama atau terdistribusi dihibridisasi dengan CUDA untuk mengimplementasikan komputasi skala besar pada kluster HPC multi-GPU. Algoritma paralel untuk CFD pada cluster multi-GPU berdasarkan MPI dan CUDA ditunjukkan dalam Algoritma 1.

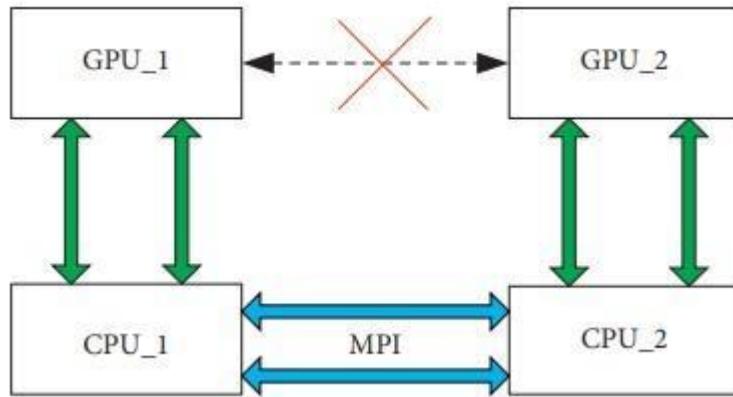
```

(1) MPI_Init (&argc, &argv);
(2) Device_Query ( );
(3) cudaMemcpy (d_a, h_a, sizeof(float)*n, cudaMemcpyHostToDevice);
(4) //Kernel execution start
(5) for i=0; i< max_step; i++
(6) Boundary_Processing_GPU<<<Block_size, Thread_size>>> ( );
(7) Time_Step_GPU<<<Block_size, Thread_size>>> ();
(8) Primitive_Variables_Exchange ();
(9) Grad_Primitive_Variabels_GPU<<<Block_size, Thread_size>>> ( );
(10) Grad_primitive_Variables_Exchange ();
(11) Flux_GPU<<<Block_size, Thread_size>>> ( );
(12) Primitive_Variables_Update_GPU<<<Block_size, Thread_size>>> ( );
(13) end for
(14) //kernel execution end
(15) cudaMemcpy (h_a, d_a, sizeof(float)*n, cudaMemcpyDeviceToHost);
(16) Flow_post-processing ();
(17) MPI_finalize ();

```

Algoritma 1. Algoritma Paralel untuk CFD di Multi GPU GPC cluster.

Program CUDA memiliki dua mode eksekusi: mode sinkron dan mode kronus asyn. Model sinkron berarti kontrol tidak kembali ke host sampai fungsi kernel saat ini dieksekusi. Mode asinkron berarti kontrol kembali ke host segera setelah fungsi kernel dimulai, oleh karena itu, host dapat memulai fungsi kernel baru dan melakukan pertukaran data secara bersamaan. Stream adalah urutan urutan perintah yang dapat dikelola oleh program CUDA untuk mengontrol paralelisme tingkat perangkat. Perintah dalam satu aliran dijalankan secara berurutan, tetapi mengalir dari perintah yang berbeda dapat dieksekusi secara paralel. Untuk itu, eksekusi bersamaan dari beberapa fungsi kernel, yaitu disebut eksekusi konkuren asinkron, dapat diimplementasikan melalui aliran. Untuk komputasi paralel besar-besaran dari CFD pada GPU, serangkaian fungsi kernel harus dieksekusi. Eksekusi bersamaan dari fungsi kernel ini dapat diimplementasikan melalui aliran. CUDA menciptakan dan menghancurkan streaming melalui fungsi cudaStreamCreate dan cudaS treamDestroy, dan sinkronisasi antar utas dapat dicapai melalui cuda Device Synchronize fungsi. Implementasi algoritme eksekusi CFD asinkron saat ini pada GPU ditunjukkan dalam Algoritma 2.



Gambar. Transfer Data Proses antar GPUs

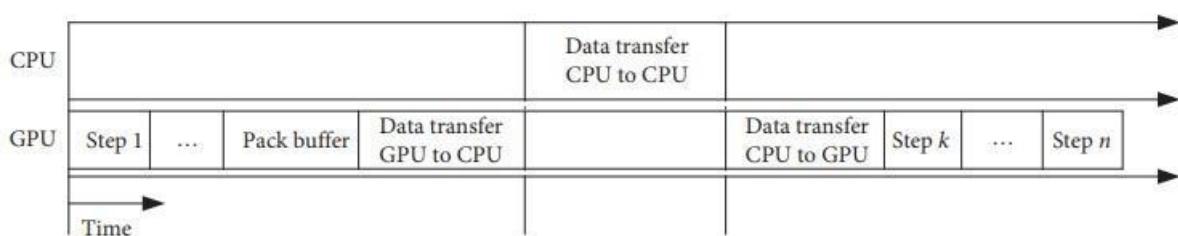
```

(1) cudaStreamCreate (&stream[j]);
(2) Boundary_Processing_GPU<<<Block_size, Thread_size, stream[0]>>>( );
(3) Time_Step_GPU<<<Block_size, Thread_size, stream[1]>>>( );
(4) Grad_Initial<<<Block_Size, Thread_Size, stream[2]>>>( );
(5) RHS_Initial<<<Block_Size, Thread_Size, stream[3]>>>( );
(6) cudaDeviceSynchronize ( );
(7) Grad_Primitive_Variables_GPU<<<Block_size, Thread_size, stream[0]>>>( );
(8) Convective_Flux_GPU<<<Block_size, Thread_size, stream[1]>>>( );
(9) cudaDeviceSynchronize ( );
(10) Viscous_Flux_GPU<<<Block_size, Thread_size, stream[0]>>>( );
(11) RHS_GPU<<<Block_size, Thread_size, stream[0]>>>( );
(12) Primitive_Variables_Update_GPU<<<Block_size, Thread_size, stream[0]>>>( );
(13) cudaDeviceSynchronize ();
(14) cudaStreamDestroy (stream[j]);

```

Algoritma 2. Algoritma eksekusi bersamaan asinkron dari CFD pada GPU.

Saat tampil komputasi paralel CFD pada kluster HPC multi-GPU, proses iterasi kernel perlu bertukar data di batas, termasuk variabel primitif dan gradiennya. variabel primitif dipilih untuk memastikan bahwa data sekecil mungkin diubah. Dalam proses ini, komunikasi CPU_CPU dan Transfer data CPU_GPU ada. Mengoptimalkan komunikasi antar GPU secara signifikan mempengaruhi kinerja sistem paralel multi-GPU. Metode tradisional adalah model komunikasi pemblokiran, seperti

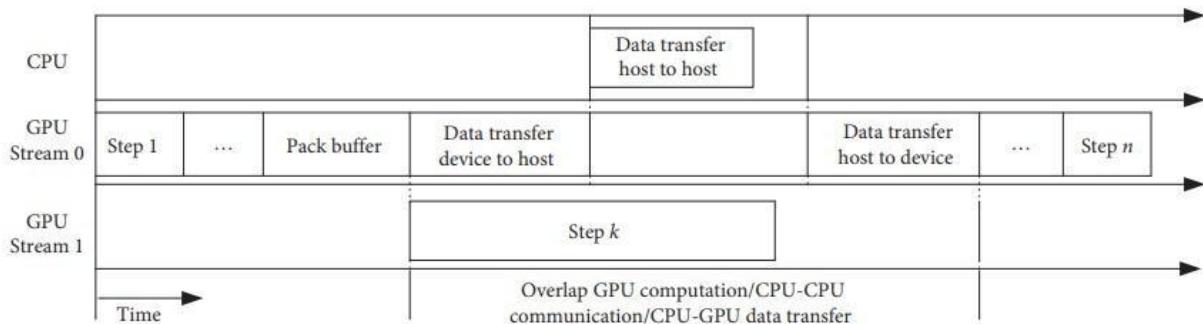


Gambar. Mode komunikasi Nonblocking

```
(1) if device_count>1 then
(2)   cudaMemcpy (h_a, d_a, sizeof(float)*n, cudaMemcpyDeviceToHost);
(3)   MPI_Bsend (*buf, int count, MPI_Datatype, int dest, int tag,MPI_COMM_WORLD);
(4)   MPI_Recv (*buf, int count, MPI_Datatype, int source, int tag,MPI_COMM_WORLD, MPI_Status *n,
cudaMemcpyHostToDevice);
(5)   cudaMemcpy (d_a, h_a, sizeof(float)*n, cudaMemcpyHostToDevice);
(6) end if
```

Algoritma 3. Algoritma Mode komunikasi Nonblocking

Memblokir mode komunikasi dengan memanggil MPI_Bsend dan MPI_Recv berfungsi untuk transmisi dan penerimaan data masing-masing. Dalam mode komunikasi pemblokiran, GPU komputasi, komunikasi CPU_CPU, dan CPU_GPU transfer data benar-benar terpisah. waktu komunikasi dalam mode komunikasi ini adalah overhead murni, yang secara serius mengurangi efisiensi paralel sistem. Mode komunikasi nonblocking melindungi waktu komunikasi dengan waktu komputasi dengan tumpang tindih komputasi dan komunikasi, seperti



Gambar. Mode komunikasi Nonblocking

```
(1) if device_count>1 then
(2)   cudaMemcpyAsync (h_a, d_a, sizeof(float)*n, cudaMemcpyDeviceToHost, stream[0]);
(3)   MPI_Isend (*buf, int count, MPI_Datatype, int dest, int tag, MPI_COMM_WORLD, MPI_Request *request);
(4)   //Primitive_Variables_Exchange;
(5)   Boundary_Processing_GPU<<<Block_size, Thread_size, stream[1]>>> ( );
(6)   Time_Step_GPU<<<Block_size, Thread_size, stream[1]>>> ( );
(7)   //Grad_Primitive_Variables_Exchange;
(8)   Convective_Flux_GPU<<<Block_size, Thread_size, stream[1]>>> ( );
(9)   MPI_Irecv (*buf, int count, MPI_Datatype, int source, int tag, MPI_COMM_WORLD, MPI_Status *status, MPI_Request
*request);
(10)  MPI_Waitall ();
(11)  cudaMemcpyAsync (d_a, h_a, sizeof(float)*n, cudaMemcpyHostToDevice, stream[0]);
(12) end if
```

Algoritma 4. Algoritma Mode komunikasi Nonblocking

Tumpang tindih di antara GPU komputasi, komunikasi CPU_CPU, dan CPU_GPU transfer data dicapai dengan membuat dua aliran CUDA. Saat menukar variabel primitif, aliran 0

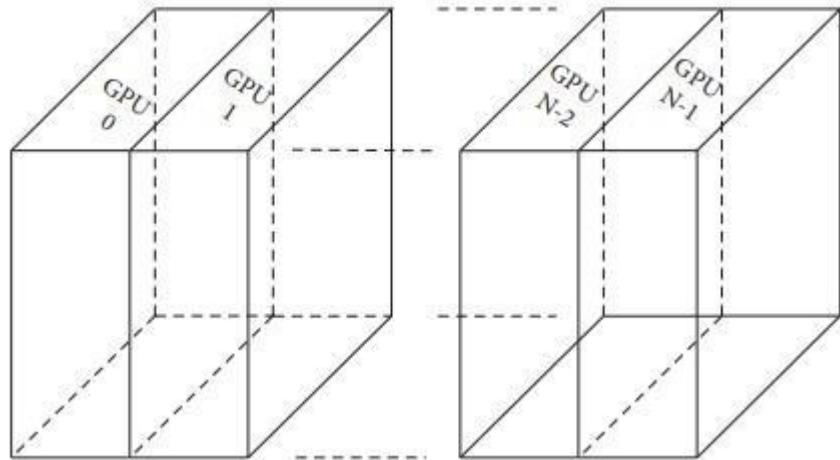
digunakan untuk transfer data CPU_GPU dan aliran 1 digunakan untuk batas pemrosesan kondisi dan perhitungan langkah waktu. Ketika gradien dari variabel primitif dipertukarkan, aliran 1 digunakan untuk menghitung fluks inviscid.

Hal ini dapat dilakukan karena perhitungan fluks yang tidak bersuara tidak tergantung pada nilai yang ditransfer di antara GPU. Pertukaran gradien ke host dapat dimulai segera setelah data dikemas dengan menggunakan stream 0. Data transmisi antara CPU diimplementasikan dengan MPI. Pada saat yang sama, fungsi convective_flux_gpu dipotong dengan menggunakan stream 1. Akhirnya, perangkat target dapat menerima data dari host dengan aliran 0. +sebelum, tumpang tindih di antara komputasi GPU, komunikasi CPU_CPU, dan transfer data CPU_GPU dapat direalisasikan. Dalam pekerjaan ini, mode komunikasi nonblocking berdasarkan multistream komputasi digunakan untuk mengoptimalkan komunikasi GPU program paralel. Mode Komunikasi Nonblocking memanggil fungsi MPI_ISEND dan MPI_IRecv untuk data transmisi dan penerimaan, masing-masing, dan MPI-fungsi WaitAll untuk menunggu penyelesaian komunikasi dan meminta status penyelesaian. Cudamemcpyasync fungsi digunakan untuk transmisi data asinkron.

Dalam komputasi paralel multi-GPU, kisi komputasi perlu dipartisi. Mempertimbangkan keseimbangan beban masalah, pekerjaan ini menggunakan dekomposisi domain 1D metode untuk memuat setiap GPU dengan kira-kira sama jumlah jaringan komputasi. 1D Domain Decompo dari metode pembagian, yang mudah diimplementasikan dan memfasilitasi penyeimbangan beban. Dikotomi Algoritma ditampilkan dalam algoritma 5, yaitu:

```
(1) for i = 1; i < partition_number; i++  
(2)   a = min_XYZ; b = max_XYZ; xyz_current = 0.5 * (min_XYZ + max_XYZ);  
(3)   while abs (dis) > 1.0exp-10 do  
(4)     dis=(current_count-average_count)/average_count;  
(5)     if dis>0.0 then  
(6)       b = xyz_current;  
(7)     else  
(8)       a = xyz_current;  
(9)     end if  
(10)    xyz_current = 0.5 * (a + b);  
(11)   end while  
(12) end for
```

Algoritma 5. Dikotomi Algoritma

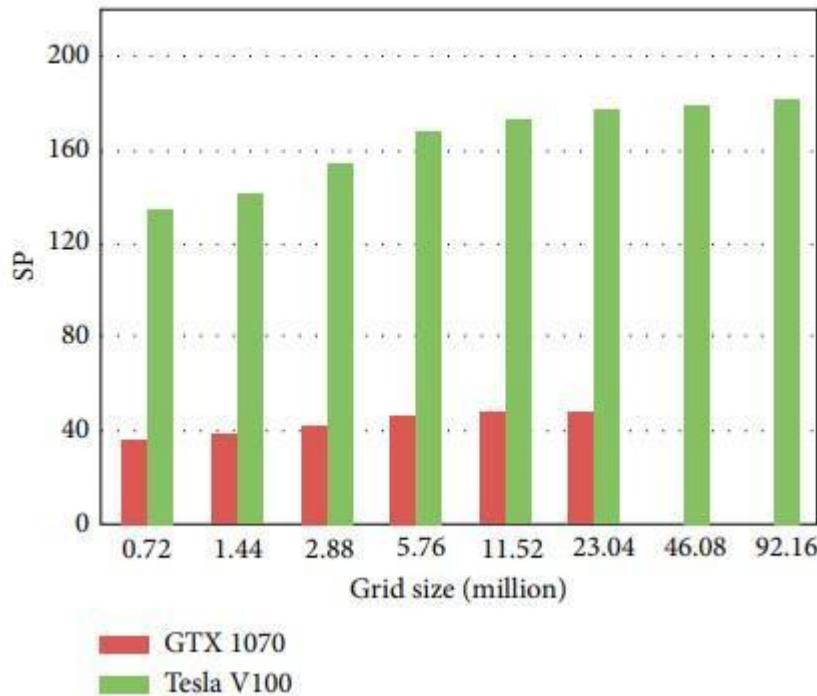


Gambar. 1D domain Decomposisi Metode

Sementara itu, akses memori yang bersatu mudah implementasikan karena penyelarasan data batas untuk meningkatkan efisiensi data batas secara efektif komunikasi. Saat komunikasi nonblocking mode digunakan, data dalam GPU dibagi menjadi tiga bagian (atas, tengah, dan bawah). Atas dan bawah bagian dari data perlu ditukar dengan perangkat lain. Transfer data bagian atas dan bawah dapat terjadi bersamaan dengan perhitungan tengah bagian.

No.	CPU (ms)	GTX 1070 (ms)	Tesla V100 (ms)
Mesh 1	567.29	15.69	4.19
Mesh 2	1,170.6	30.61	8.27
Mesh 3	2,619.41	62.75	16.9
Mesh 4	5,605.29	120.25	33.18
Mesh 5	11,258.58	236.29	64.88
Mesh 6	22,850.69	476.12	128.98
Mesh 7	46,211.38	—	256.72
Mesh 8	93,322.76	—	512.44

Tabel. Runtime untuk satu CPU dan single GPU



Gambar. Speedup dari satu GPU untuk jumlah sel grid yang berbeda

Gambar diatas menunjukkan bahwa percepatan satu GPU meningkat dengan meningkatnya ukuran grid. Untuk Tesla V100 GPU, speedup mencapai 135,39 untuk mesh 1 dan 182,11 untuk mesh 8. GPU Tesla V100 memiliki performa yang jauh lebih besar speedup dari GTX 1070 GPU. Paralelisasi GPU dapat sangat meningkatkan efisiensi komputasi dibandingkan dengan komputasi paralel berbasis CPU. Gambar percepatan dan efisiensi diatas itu menunjukkan bahwa GPU GTX 1070 tidak mampu membayar sejumlah besar perhitungan karena keterbatasan memori perangkat. Untuk kode GPU, memori perangkat 1 GB dapat memuat sekitar 3 juta sel jaringan. Dengan bertambahnya ukuran grid, percepatan kode GPU meningkat secara bertahap alasannya adalah bahwa dengan bertambahnya ukuran grid, proporsi kernel eksekusi meningkat dengan pengaturan data dan komunikasi. Sementara itu, laju pertumbuhan percepatan adalah berangsur-angsur berkurang karena keterbatasan jumlah dari inti SM dan CUDA.

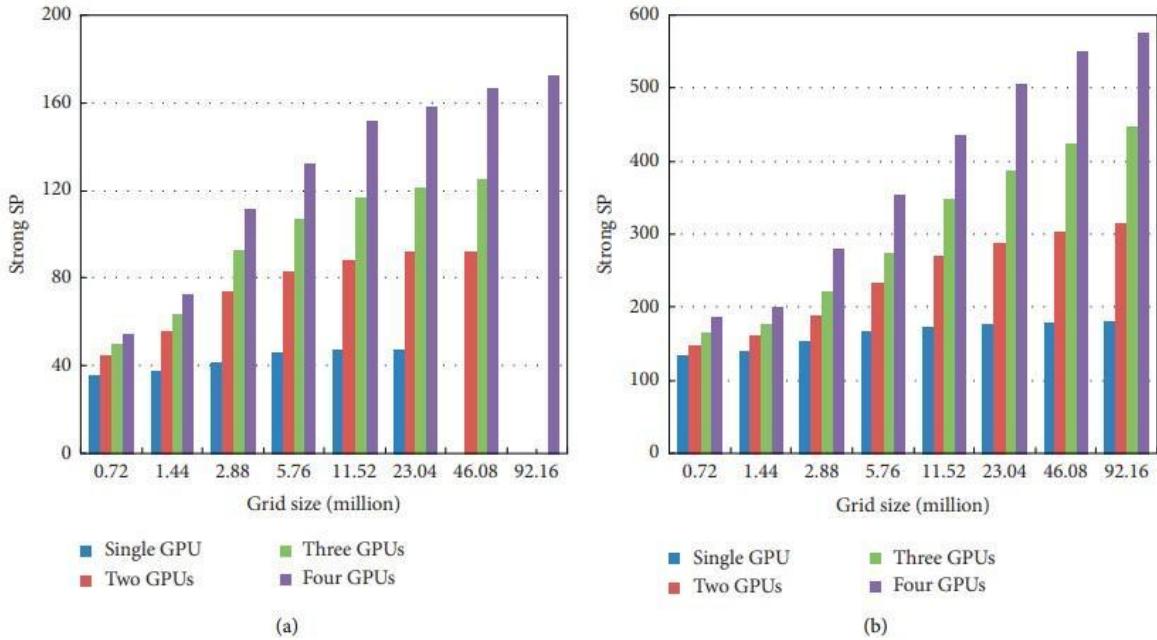
Pada bagian ini, mode komunikasi pemblokiran digunakan untuk mempelajari skalabilitas kode GPU. Di sini, kinerja cluster HPC multi GPU GTX 1070 dan Tesla V100 dibahas. Tes penskalaan yang kuat dilakukan untuk jerat 1 hingga 8 pada cluster HPC multi-GPU.

No.	Two GPUs (ms)	Three GPUs (ms)	Four GPUs (ms)
Mesh 1	12.62	11.27	10.38
Mesh 2	20.86	18.19	16.01
Mesh 3	35.56	28.19	23.38
Mesh 4	67.55	52.43	42.33
Mesh 5	126.93	96.24	77.15
Mesh 6	252.01	187.84	144.43
Mesh 7	499.02	369.68	276.86
Mesh 8	—	—	540.73

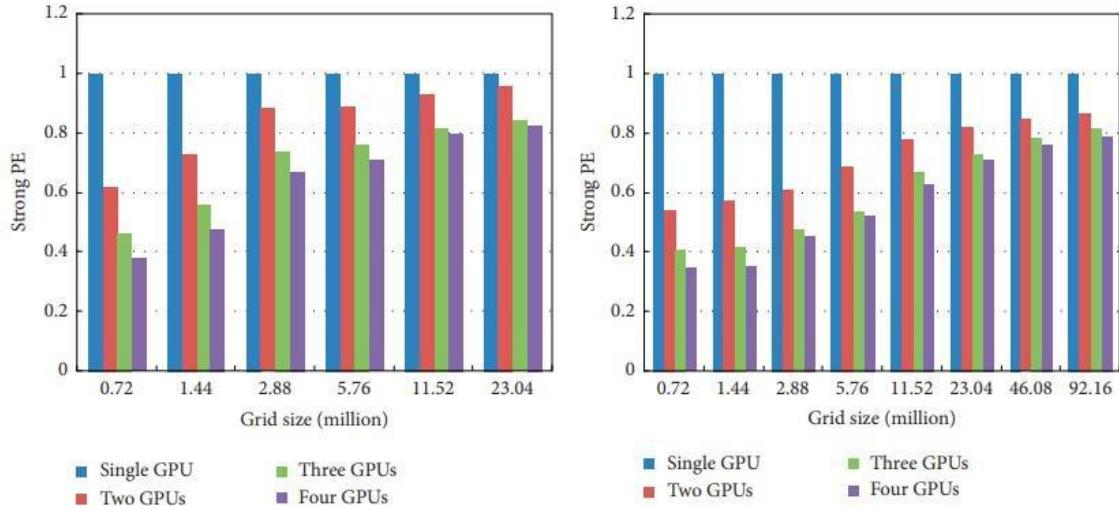
Tabel. runtime untuk kluster multi-GPU GTX 1070.

No.	Two GPUs (ms)	Three GPUs (ms)	Four GPUs (ms)
Mesh 1	3.85	3.41	3.01
Mesh 2	7.16	6.58	5.86
Mesh 3	13.78	11.82	9.31
Mesh 4	23.99	20.43	15.8
Mesh 5	41.59	32.22	25.71
Mesh 6	78.69	58.81	45.16
Mesh 7	151.78	108.85	83.92
Mesh 8	295.13	208.72	161.88

Tabel. runtime untuk kluster Tesla V100 multi-GPU

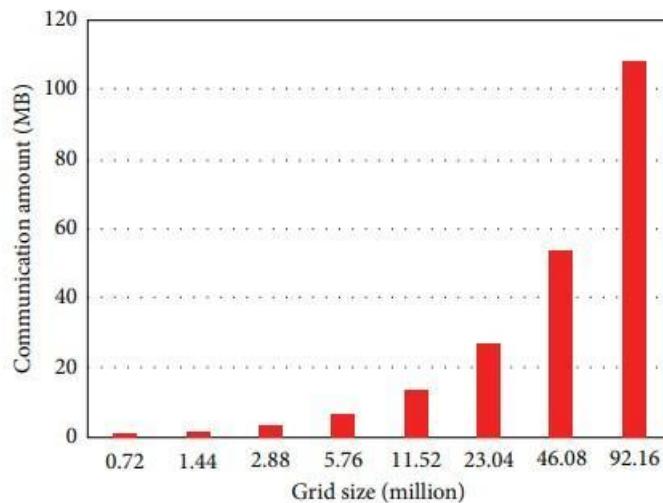


Gambar. percepatan yang kuat dari cluster multi-GPU. (a). GTX 1070 (b). Tesla V100. Gambar diatas menunjukkan bahwa percepatan yang kuat ditampilkan untuk ukuran grid yang berbeda. Terbukti, ukuran grid yang besar bisa mencapai kecepatan tinggi. Untuk GTX 1070 dan Tesla V100 multi-GPU cluster, percepatan empat GPU mencapai 172,59 dan 576,49, masing-masing. nilai ini jauh lebih besar daripada speedups dicapai oleh satu GPU. + kami, tingkat tinggi yang kuat kinerja penskalaan dipertahankan. Paralelisasi multi-GPU dapat sangat meningkatkan efisiensi komputasi dengan peningkatan ukuran grid. Namun, paralel multi-GPU komputasi tidak menunjukkan keuntungan yang jelas ketika grid ukurannya kecil dapat dijelaskan oleh fakta bahwa kerabatnya bobot pertukaran data berbanding terbalik dengan grid ukuran. GPU dikhkususkan untuk komputasi intensif, sangat komputasi paralel.



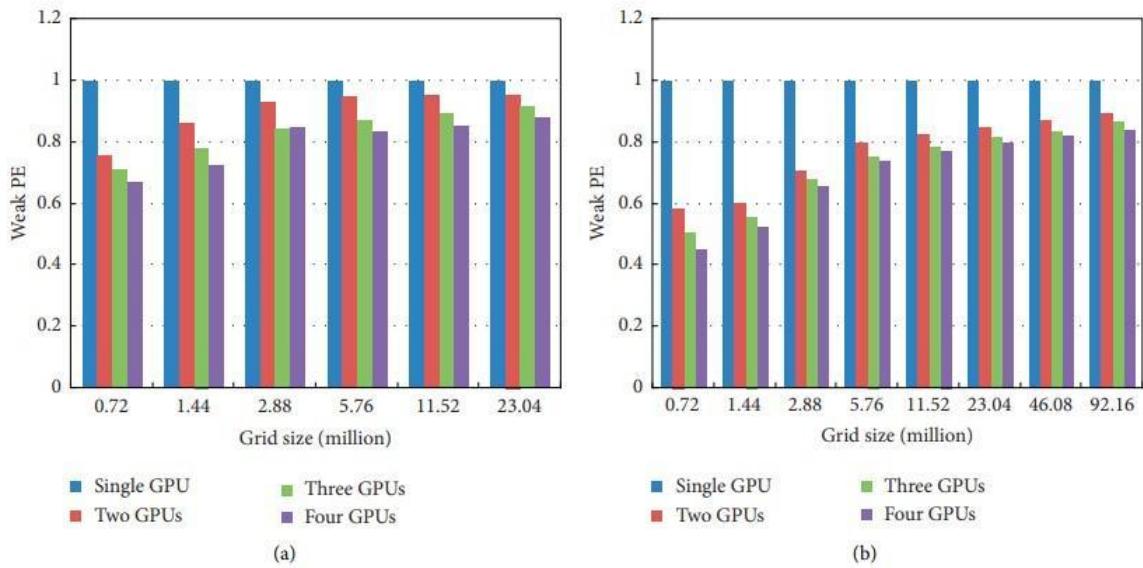
Gambar. efisiensi paralel yang kuat dari cluster multi-GPU. (a). GTX 1070 (b). Tesla V100

Efisiensi paralel yang kuat ditunjukkan untuk ukuran grid yang berbeda hasilnya konsisten dengan hukum perubahan percepatan; yaitu paralel efisiensi meningkat dengan peningkatan ukuran grid, dan kinerja efisiensi paralel yang kuat dari multi cluster GTX 1070 sedikit lebih baik daripada GPU Tesla V100. Untuk mesh 8, efisiensi paralel yang kuat dari empat Tesla V100 GPU mendekati 80%. Selain itu, jumlah GPU yang lebih banyak menunjukkan efisiensi paralel yang lebih rendah karena peningkatan jumlah transfer data.



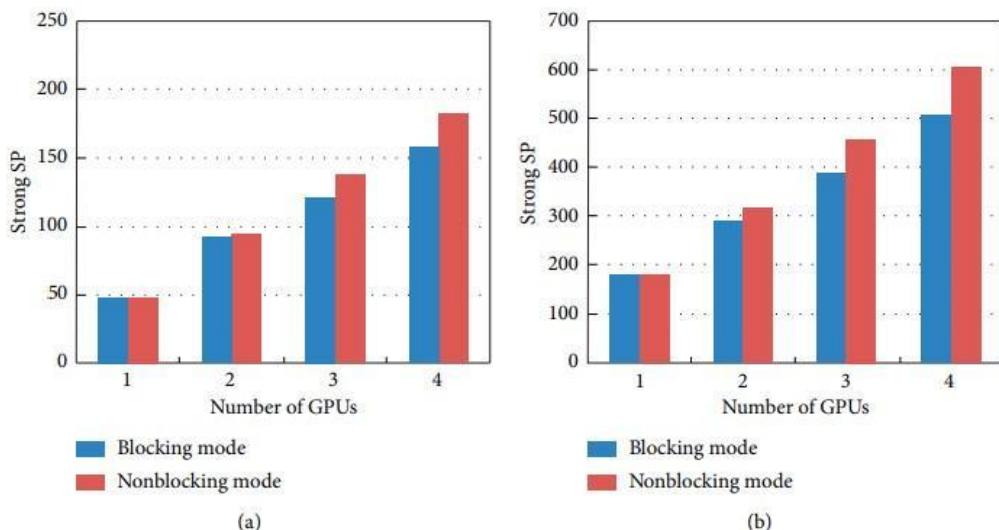
Gambar. jumlah komunikasi memori.

Gambar diatas menunjukkan jumlah komunikasi memori untuk komputasi paralel dengan empat GPU. Saat ukuran kisi meningkat, jumlah memori komunikasi meningkat secara proporsional.

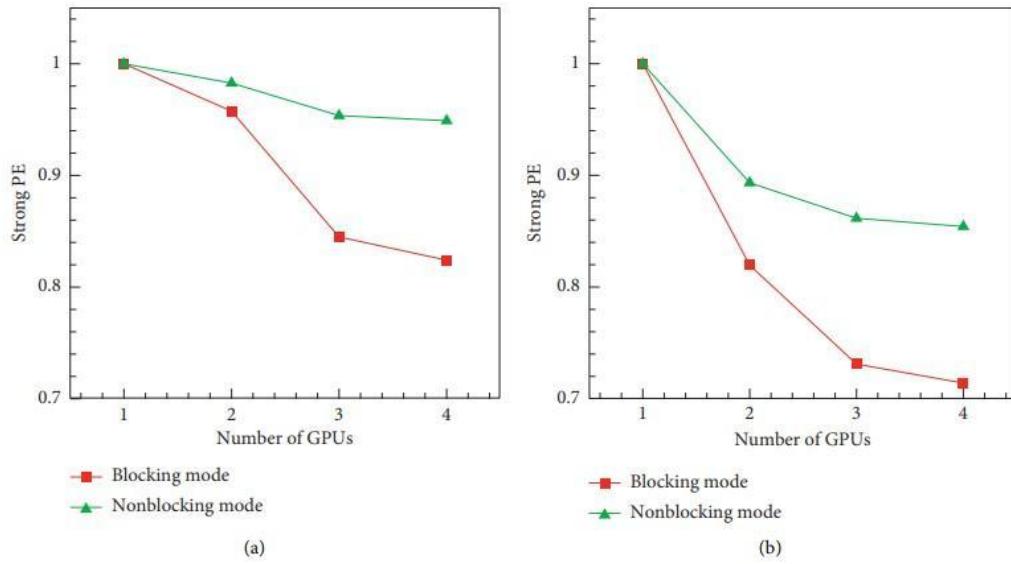


Gambar. efisiensi paralel yang lemah dari cluster multi-GPU. (a). GTX 1070 (b). Tesla V100

Tes penskalaan yang lemah untuk efisiensi paralel ditunjukkan pada Gambar diatas, dan ukuran grid yang dimuat pada setiap blok tetap ada konstan. Seperti yang diharapkan, efisiensi paralel menurun sebagai jumlah GPU meningkat karena jumlah data pertukaran meningkat dengan peningkatan jumlah GPU. Untuk ukuran grid mesh 6 pada setiap GPU GTX 1070 dan Tesla V100 GPU, efisiensi paralel lemah dari empat GPU masing-masing dapat mencapai 88,05% dan 79,68% Sehingga gelar tinggi kinerja penskalaan yang lemah dipertahankan.

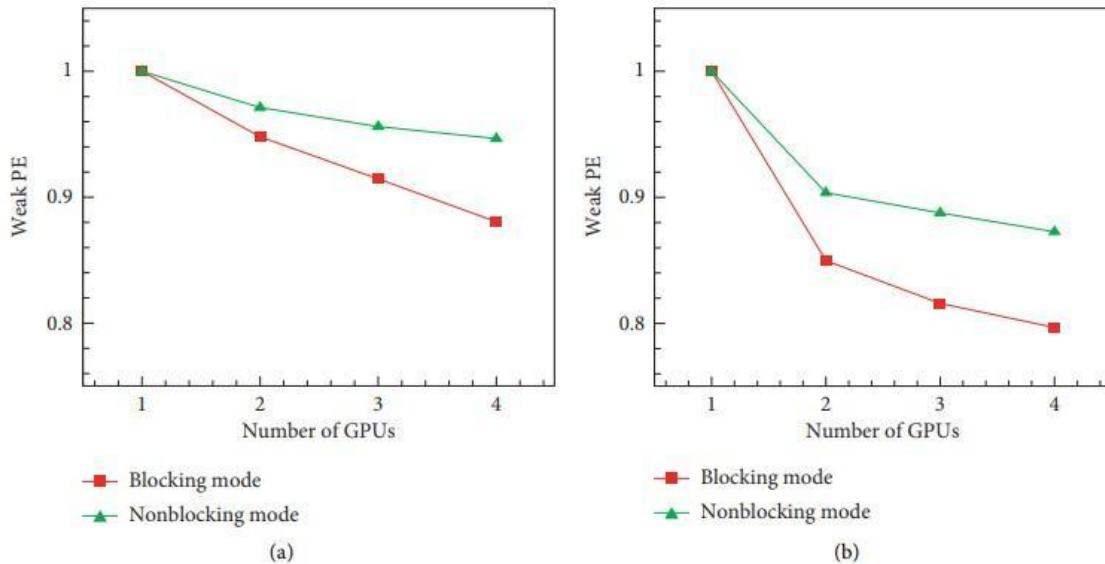


Gambar. percepatan yang kuat dari cluster multi-GPU dengan mode komunikasi yang berbeda (a). GTX 1070 (b). Tesla V100



Gambar. efisiensi paralel yang kuat dari cluster multi-GPU dengan mode komunikasi yang berbeda. (a). GTX 1070 (b). Tesla V100

Dua gambar diatas menunjukkan kinerja skalabilitas yang kuat dari mode komunikasi nonblocking dibandingkan dengan mode komunikasi pemblokiran.



Gambar. efisiensi paralel yang lemah dari cluster multi-GPU dengan mode komunikasi yang berbeda. (a). GTX 1070 (b). Tesla V100

Gambar diatas menunjukkan efisiensi paralel lemah dari dua metode. Hasil menunjukkan bahwa mode komunikasi non blocking dapat melindungi waktu komunikasi dengan waktu komputasi dengan tumpang tindih komunikasi dan komputasi. Untuk empat GPU, percepatan kuat GTX 1070 dan Tesla V100 multi-GPU masing-masing meningkat 15,15% dan 19,63%. Selain itu,

efisiensi paralel yang lemah tetap di atas 87% dengan mode komunikasi non blocking. Selain itu, Paralelisasi berbasis GPU dengan GPU Tesla V100 dapat menunjukkan peningkatan kinerja yang lebih baik daripada GTX 1070 cluster HPC multi-GPU. Hasilnya adalah karena kerabatnya bobot waktu komunikasi GPU Tesla V100 adalah lebih tinggi.

Dari penjelasan diatas dapat disimpulkan bahwa algoritma paralel hibrida MPI dan CUDA. Untuk aplikasi CFD pada cluster HPC multi-GPU telah diajukan dan metode optimasi telah diadopsi mencapai tingkat pemanfaatan paralelisme tingkat tertinggi dan throughput memori maksimum. Dalam penelitian ini, blok utas ukuran 256 digunakan. Jumlah blok utas diturunkan ditambah oleh skala beban kerja untuk memastikan bahwa setiap utas dimuat dengan perhitungan sel grid. Untuk satu GPU implementasi, dua jenis perangkat telah dibahas. Dalam penelitian ini diperoleh rasio akselerasi lebih dari 36 kali, yang menunjukkan bahwa paralelisasi GPU dapat sangat meningkatkan efisiensi komputasi dibandingkan dengan paralel berbasis CPU komputasi. Sementara itu, ukuran grid besar dapat mencapai tinggi speedup karena peningkatan proporsi kernel eksekusi dibandingkan dengan pengaturan data dan komunikasi. Speedup dari empat GPU mencapai 172.59 dan 576.49 untuk masing-masing kluster HPC Multi-GPU GTX 1070 dan Tesla V100. paralel yang kuat dan lemah efisiensi dipertahankan pada tingkat tinggi saat ukuran grid pada nilai yang besar. Algoritma paralel memiliki kuat yang baik dan skalabilitas yang lemah. Mode komunikasi non blocking telah diusulkan untuk sepenuhnya dalam tumpang tindih komputasi GPU, Komunikasi CPU_CPU, dan transfer data CPU_GPU. Untuk empat GPU, speedup kuat GTX 1070 dan Tesla V100 Multi-GPU meningkat sebesar 15,15% dan 19,63%, secara respirasi. Selain itu, efisiensi paralel yang lemah tetap di atas 87% dengan mode komunikasi non blocking.

Penutup Kesimpulan

1. Komputasi awan (cloud computing) yang salah satunya mempunyai sifat elasticity di dalamnya, menawarkan solusi keterbatasan sumber daya ini. Sumber daya yang terbatas misalkan dalam hal processor, RAM atau storage, dapat digabungkan dengan sumber daya yang dimiliki public cloud provider yang tersedia di market. Sehingga penggabungan 2 sumber daya ini, private cloud dan public cloud, dimana penggabungan ini dapat menjadi solusi untuk dapat mengimplementasikan analisis big data yang dapat diterapkan untuk analisis berbagai macam bidang.
2. MPI dan OpenMP adalah dua pemrograman aplikasi antarmuka yang banyak digunakan untuk menjalankan kode paralel pada platform multi-GPU. OpenMP dapat digunakan dalam node untuk paralelisasi berbutir halus menggunakan memori bersama, dan MPI bekerja pada memori yang dibagikan dan didistribusikan dan banyak digunakan untuk komputasi paralel besar-besaran. Secara umum, MPI

menunjukkan penurunan kinerja dibandingkan dengan OpenMP tetapi relatif mudah dilakukan pada berbagai jenis perangkat keras dan memiliki skalabilitas yang baik.

3. Unit pemrosesan grafis (GPU) memiliki kemampuan floating-point yang kuat dan bandwidth memori yang tinggi dalam paralelisme data dan telah banyak digunakan dalam komputasi kinerja tinggi (HPC). Compute unified device architecture (CUDA) digunakan sebagai paralel platform komputasi dan model pemrograman untuk GPU untuk mengurangi kerumitan pemrograman. GPU dapat diprogram menjadi populer dalam aplikasi dinamika fluida komputasi (CFD).
4. Program CUDA memiliki dua mode eksekusi: mode sinkron dan mode kronus asyn. Model sinkron berarti kontrol tidak kembali ke host sampai fungsi kernel saat ini dieksekusi. Mode asinkron berarti kontrol kembali ke host segera setelah fungsi kernel dimulai, oleh karena itu, host dapat memulai fungsi kernel baru dan melakukan pertukaran data secara bersamaan.
5. Stream adalah urutan urutan perintah yang dapat dikelola oleh program CUDA untuk mengontrol paralelisme tingkat perangkat. Perintah dalam satu aliran dijalankan secara berurutan, tetapi mengalir dari perintah yang berbeda dapat dieksekusi secara paralel. Untuk itu, eksekusi bersamaan dari beberapa fungsi kernel, yaitu disebut eksekusi konkuren asinkron, dapat diimplementasikan melalui aliran.
6. Untuk komputasi paralel besar-besaran dari CFD pada GPU, serangkaian fungsi kernel harus dieksekusi. Eksekusi bersamaan dari fungsi kernel ini dapat diimplementasikan melalui aliran. CUDA menciptakan dan menghancurkan streaming melalui fungsi cudaStreamCreate dan cudaS team Destroy, dan sinkronisasi antar utas dapat dicapai melalui fungsi cuda Device Synchronize.

Daftar Pustaka

Abuin JM, Lopes N, Ferreira L, Pena TF, & Schmidt (2020) Big Data in metagenomics: Apache

Spark vs MPI. *PLoS ONE* 15(10): e0239741

Aleem, Muhammad. 2015. A Java-based Programming and Execution Environment for Manycore Parallel Computers. Proquest LLC is an Ann Arbor, Michigan, USA. diakses dari
http://www.researchgate.net/publication/302956570_A_Java-based_Programming_and_Execution_Environment_for_Many-core_Parallel_Computers tanggal 21 September 2022

Ardianti, Dwi., Zahrah, F., Yusup,M., Handayani, P., Ariztian,R. (2014).SISTEM TERDISTRIBUSI" RINGAKASAN MATERI PERTEMUAN 1 S/D 11.

Bak, Seonmyeong, et. al. 2021. *OpenMP Application Experiences: Porting to Accelerating Nodes*.Parallel Computing 109 (2021).

Bassil Y (2019) Memory-Based MultiProcessing Method For Big data Computation.

International Journal of Advanced Research and Publication 3(3): 141-146

Chew, Weng Cho. 2013.Overview of Large-Scale Computing: The Past, the Present, and the Future. *Proceedings of the IEEE* 101 (2): 227-241

Cinderatama, T.A., Yunhasnawa, Y., Alhamri, R.Z. (2018). Desain Dan Implementasi Hybrid Cloud Computing Sebagai Infrastruktur Untuk Analisis Big Data Menggunakan Analytic Hierarchy Process(AHP). *Techno.COM*, Vol. 17, No. 4: 404-414.

Fei Hu, Chaowei Yang, John L. Schnase, Daniel Q. Duffy, Mengchao Xu, Michael K. Bowen, Tsengdar Lee, Weiwei Song, (2018) ClimateSpark: An in-memory distributed computing framework for big climate data analytics, *Computers & Geosciences*, 115: 154-166

Junjie Hou, Yongxin Zhu, Sen Du, & Shijin Song (2018) Design and implementation of reconfigurable acceleration for in-memory distributed big data computing, *Future*

Generation Computer System, 92: 68-75

Lai, Jianqi., Yu, Hang., Tian, Zhengyu., Li, Hua. 2020. Hybrid MPI and CUDA Parallelization for CFD Applications on Multi-GPU HPC Clusters. Hindawi Scientific Programming.

Narayanan S, Samuel P, & Chacko M (2020) Improving prediction with enhanced Distributed Memory-based Resilient Dataset Filter. *Journal of Big Data*, 7(13)

Rahman MdA, Hossen J, Sultana A, Mamun AA, & Aziz NAA. (2021) A smart method for spark using neural network for big data. *International Journal of Electrical and Computer Engineering (IJECE)* 11(3): 2525-2534

Link ppt:

https://univindonesia-my.sharepoint.com/:p/g/personal/elizabeth_lilies_office_ui_ac_id/EceAPk/qviTdNkeg27gY4Cw0B16k-DhNzj98HVnxatkm2mQ

Link video part 1: <https://youtu.be/2zgqm-EAFQo>

Link video part 2: <https://youtu.be/fTAIPw0p6gk>

BAB 5

Analisis Algoritma Paralel

Cintya Kusuma Mahadhika, Eka Dwi Agustina Ginting, Elizabeth Lilies Megawati

Abstrak

Dalam ilmu komputer, algoritma paralel, berbeda dengan algoritma sekuensial (atau serial) tradisional, adalah algoritma yang dapat dieksekusi secara berurutan oleh beberapa proses berbeda dan kemudian digabungkan kembali menjadi satu. Pengguna akan mendapatkan hasil yang tepat. Algoritma paralel sendiri memiliki beberapa teknik perancangan maupun analisis kompleksitas. Teknik perancangan algoritma paralel sendiri meliputi desain teknik dan desain *compiler*, yang masing-masing memiliki kategorinya sendiri.

Kata Kunci: algoritma paralel, analisis algoritma paralel, teknik perancangan

Pendahuluan

Dalam ilmu komputer, algoritma paralel, berbeda dengan algoritma sekuensial (atau serial) tradisional, adalah algoritma yang dapat dieksekusi secara berurutan oleh beberapa proses berbeda dan kemudian digabungkan kembali menjadi satu. Pengguna akan mendapatkan hasil yang tepat. Di sisi lain, sebagian besar algoritma yang tersedia untuk menghitung $\Pi(\pi)$ tidak dapat dengan mudah dibagi menjadi bagian paralel. Pengguna memerlukan hasil dari langkah sebelumnya untuk melanjutkan ke langkah berikutnya secara efektif. Masalah seperti ini disebut masalah serial. Metode numerik iteratif, seperti metode Newton atau masalah tiga benda, juga merupakan algoritma yang bersifat serial. Beberapa masalah sangat sulit untuk diparalelkan meskipun bersifat rekursif. Contohnya adalah pencarian mendalam pertama dari grafik. Algoritma paralel berharga karena peningkatan yang signifikan dalam sistem multiprosesor dan munculnya prosesor multicore. Secara umum, lebih mudah membuat komputer dengan satu prosesor cepat daripada membuat komputer dengan banyak prosesor lambat dengan kinerja yang sama. Namun, kecepatan prosesor terutama didorong oleh sirkuit menyusut, dan prosesor modern mendorong ukuran fisik dan batas termal. Resistor ganda membalikkan persamaan, membuat *multi processing* dapat dilakukan bahkan dalam sistem kecil.

Pembahasan

1. ANALISIS ALGORITMA PARALEL

1.1. Pengertian Algoritma Paralel

Algoritma adalah bagian penting dari struktur data. Struktur data diimplementasikan menggunakan algoritma. Algoritma adalah prosedur yang dapat ditulis sebagai fungsi atau program dalam C atau bahasa lain. Algoritma menentukan dengan tepat bagaimana data akan diproses (

Algoritma paralel adalah teknik pemecahan masalah yang dirancang untuk komputer paralel. Simulasi diterapkan dalam perkalian dengan matriks dan matriks dengan vektor

dalam algoritma dan metode PASCAL. Algoritma paralel dirancang untuk diproses dalam komputer SM-SISD (Shared Memory, Single Instruction Stream - Single Data Stream) yang terhubung dengan model kubus dan pohon. Fokus pada sistem komputer yang digunakan saat ini masih berupa komputer tunggal, karena memori yang digunakan adalah multi address virtual memory (dadox, 2013).

1.2. Teknik Dasar Algoritma Paralel

Dalam beberapa kasus, algoritma sekuensial dapat dengan mudah disesuaikan dengan lingkungan paralel. Namun, dalam banyak kasus, masalah komputasi harus dianalisis ulang dan algoritma paralel baru dibuat. Beberapa studi telah dilakukan pada desain algoritma paralel untuk masalah praktis seperti pengurutan, pemrosesan gerak, penyelesaian persamaan linier, penyelesaian persamaan diferensial, dan simulasi (Purbasari, 2005).

Teknik pembangunan algoritma paralel dapat dibedakan sebagai berikut (Purbasari, 2005).

a. Paralelisme Data

Teknik paralelisme data adalah teknik yang paling banyak digunakan dalam program paralel. Teknik tersebut muncul dari penelitian bahwa aplikasi utama komputasi paralel adalah di bidang sains dan teknologi, yang biasanya melibatkan array multidimensi yang sangat besar. Dalam program sekuensial normal, array ini dimanipulasi menggunakan loop bersarang untuk mendapatkan hasilnya. Sebagian besar program paralel dibangun dengan mengatur ulang algoritma sekuensial sehingga loop bersarang dapat dieksekusi secara paralel. Paralelisme data menunjukkan bahwa basis data digunakan sebagai dasar untuk membuat fungsi paralel, dimana berbagai bagian basis data diproses secara paralel. Dengan kata lain, paralelisme program terdiri dari penerapan operasi yang sama ke bagian tabel data yang berbeda. Prinsip paralelisme data ini berlaku untuk multiprosesor dan pemrograman multi komputer (Purbasari, 2005).

b. Partisi Data

Ini adalah teknik paralelisme data khusus, di mana data didistribusikan pada memori lokal beberapa komputer. Setelah itu, semua data diverifikasi oleh proses paralel. Suatu proses harus berada di penyimpanan lokal yang sama dengan partisi data agar proses dapat mengakses data secara lokal. Untuk kinerja yang baik, setiap proses harus mengetahui variabel dan data lokalnya sendiri. Ketika suatu proses perlu mengakses data yang terdapat dalam memori jarak jauh, itu dapat dilakukan melalui jaringan pengiriman pesan yang menghubungkan prosesor. Karena komunikasi antar proses menimbulkan penundaan, penyampaian pesan harus dilakukan pada frekuensi yang relatif rendah. Dari sini dapat disimpulkan bahwa tujuan pemisahan data adalah untuk mengurangi delay yang diakibatkan oleh pengiriman pesan antar proses. Algoritma paralel mengatur setiap proses untuk melakukan perhitungan pada data lokalnya sendiri (Purbasari, 2005).

c. Algoritma Relaksasi

Dengan algoritma ini, setiap proses tidak memerlukan sinkronisasi dan komunikasi antar proses. Meskipun pemroses menggunakan data yang sama, setiap pemroses dapat melakukan kalkulasinya sendiri secara independen dari data antara yang dihasilkan oleh

proses lain. Contoh algoritma relaksasi adalah algoritma perkalian matriks, pengurutan menggunakan metode rank sort, dll. (Purbasari, 2005).

d. Paralelisme Sinkron

Aplikasi praktis komputasi paralel adalah masalah dengan array multidimensi yang sangat besar. Masalah ini memberikan peluang yang baik untuk paralelisme data karena elemen tabel yang berbeda dapat diproses secara paralel. Teknik perhitungan numerik pada tabel ini biasanya bersifat iteratif, dimana setiap iterasi mempengaruhi iterasi selanjutnya menuju solusi akhir. Misalnya, memecahkan persamaan numerik dalam sistem besar. Secara umum, setiap iterasi menggunakan informasi yang dihasilkan oleh iterasi sebelumnya, dan akhirnya program sampai pada solusi dengan akurasi yang dibutuhkan. Algoritma iteratif ini memiliki kemungkinan paralelisme di setiap iterasi. Beberapa proses paralel dapat dikonfigurasi untuk berjalan di berbagai bagian array. Namun setelah setiap iterasi, proses harus disinkronkan karena hasil yang dihasilkan oleh satu proses digunakan oleh proses lain pada iterasi berikutnya. Jenis teknik pemrograman paralel ini disebut paralel sinkron. Contohnya adalah perhitungan numerik dengan menggunakan metode eliminasi Gaussian. Dalam paralelisme sinkron, struktur umum suatu program biasanya terdiri dari loop UNTUK SEMUA, yang membentuk sejumlah besar proses paralel untuk bekerja di berbagai bagian tabel data. Setiap proses diulang dan disinkronkan pada akhir iterasi. Sinkronisasi ini bertujuan untuk memastikan bahwa semua proses telah menyelesaikan iterasi saat ini sebelum proses memulai iterasi berikutnya. (Purbasari, 2005).

e. Komputasi Pipeline

Dengan perhitungan pipa geser, data mengalir melalui seluruh struktur proses, dengan setiap proses membentuk tahapan tertentu dari keseluruhan perhitungan. Algoritma ini dapat bekerja dengan baik pada banyak komputer karena terdapat aliran data dan tidak memerlukan banyak akses ke informasi yang dibagikan. Contoh penghitungan pipa adalah teknik substitusi mundur, yang merupakan bagian dari metode eliminasi.

Saat merancang algoritma paralel, perlu dipertimbangkan hal-hal yang dapat memperlambat kinerja program paralel, yaitu (Purbasari, 2005):

a. Memory Contention

Eksekusi prosesor tertunda ketika prosesor menunggu hak akses ke sel memori yang sedang dipergunakan oleh prosesor lain. Problem ini muncul pada arsitektur multiprosesor dengan memori bersama.

b. Excessive Sequential Code

Pada beberapa algoritma paralel, akan terdapat kode sekuensial murni dimana tipe tertentu dari operasi pusat dibentuk, seperti misalnya pada proses inisialisasi. Dalam beberapa algoritma, kode sekuensial ini dapat mengurangi keseluruhan speedup yang dapat dicapai.

c. Process Creation Time

Pembentukan proses paralel akan membutuhkan waktu eksekusi. Jika proses yang dibentuk relative berjalan dalam waktu yang relatif lebih singkat dibandingkan dengan

waktu pembentukan proses itu sendiri, maka overhead pembuatan akan lebih besar dibandingkan dengan waktu eksekusi paralel algoritma.

d. Communication Delay

Overhead ini muncul hanya pada multikomputer. Hal ini disebabkan karena interaksi antar prosesor melalui jaringan komunikasi. Dalam beberapa kasus, komunikasi antar dua prosesor mungkin melibatkan beberapa prosesor antara dalam jaringan komunikasi. Jumlah waktu tunda komunikasi dapat menurunkan kinerja beberapa algoritma paralel.

e. Synchronization Delay

Ketika proses paralel disinkronkan, berarti bahwa suatu proses akan harus menunggu proses lainnya. Dalam beberapa program paralel, jumlah waktu tunda ini dapat menyebabkan bottleneck dan mengurangi speedup keseluruhan.

f. Load Imbalance

Dalam beberapa program paralel, tugas komputasi dibangun secara dinamis dan tidak dapat diperkirakan sebelumnya. Karena itu harus selalu ditugaskan ke prosesor-prosesor sejalan dengan pembangunan tugas tersebut. Hal ini dapat menyebabkan suatu prosesor tidak bekerja (idle), sementara prosesor lainnya tidak dapat mengerjakan task yang ditugaskannya.

1.3. Contoh Algoritma Paralel

a. Komputasi Paralel

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer independen secara bersamaan (dadox, 2013).

b. Paralel Virtual Machine (PVM)

Paralel Virtual Machine (PVM) adalah paket software yang mendukung pengiriman pesan untuk komputasi paralel antar komputer. PVM dapat berjalan diberbagai macam variasi UNIX ataupun windows dan telah portable untuk banyak arsitektur seperti PC, workstation, multiprocessor dan superkomputer (dadox, 2013).

Komputasi Paralel dengan Parallel Virtual Machine (PVM) (Purbasari, 2005)

Komputasi paralel adalah teknik di mana perhitungan dilakukan secara bersamaan menggunakan beberapa komputer independen secara bersamaan. Ini biasanya diperlukan ketika kapasitas yang dibutuhkan sangat besar, atau karena perlu adanya pemrosesan data dalam jumlah besar (di bidang keuangan, bioinformatika, dll.) atau karena banyak proses komputasi yang diperlukan. Menggunakan parallel processing saat ini menjadi pilihan yang cukup handal untuk mengolah data yang besar dan banyak, dibandingkan membeli superkomputer yang harganya sangat mahal, menggunakan parallel processing menjadi pilihan yang sangat cocok untuk diproses. tanggal-tanggal ini.

Aspek keamanan merupakan aspek penting dari parallel processing ini dalam sistem komputer, karena sistem banyak berhubungan dengan hak akses data, hak pengguna, keamanan, keamanan jaringan terhadap serangan seseorang atau bahkan virus, yang membuat kegiatan ini menjadi sulit. sistem perhitungan

Komputasi paralel adalah eksekusi kalkulasi komputer menggunakan dua atau lebih CPU/prosesor pada komputer yang sama atau pada komputer yang berbeda, memecah setiap instruksi menjadi beberapa instruksi, yang kemudian dikirim ke prosesor yang terlibat dalam kalkulasi dan dieksekusi secara bersamaan. Pada proses pemisahan perhitungan dilakukan oleh software yang berperan untuk mengatur perhitungan. Dokumen ini menggunakan teknologi mesin virtual paralel (PVM). Sistem komputasi paralel memiliki banyak unit pemrosesan dan banyak unit memori. Ada dua teknik berbeda untuk mengakses data dalam unit penyimpanan, yaitu pengalamanan memori bersama dan pengiriman pesan. Bergantung pada bagaimana memori ini diatur, komputer paralel dibagi menjadi mesin paralel memori bersama dan mesin paralel memori terdistribusi.

Prosesor dan memori mesin paralel ini dapat dihubungkan (linked) secara statis maupun dinamis. Koneksi statis sering digunakan oleh sistem memori terdistribusi. Peering langsung digunakan untuk menghubungkan semua prosesor. Tautan dinamis biasanya menggunakan sakelar untuk menghubungkan prosesor dan memori. Komunikasi data dalam sistem memori paralel terdistribusi membutuhkan dukungan transfer data. Contoh alat yang saat ini banyak digunakan oleh sistem seperti komputer jaringan adalah standar PVM (Parallel Virtual Machine), yang beroperasi pada lapisan komunikasi TCP/IP. Standar ini membutuhkan kemampuan akses jarak jauh untuk menjalankan program pada setiap unit prosesor. Salah satu protokol yang digunakan dalam komputasi paralel adalah Network File System (NFS). NFS adalah protokol yang dapat berbagi sumber daya melalui jaringan. NFS menjadi independen dari jenis mesin, jenis sistem operasi, dan jenis protokol transfer yang digunakan. Ini dilakukan dengan menggunakan RPC. NFS memungkinkan pengguna dengan izin untuk mengakses file yang terletak di host jarak jauh serta mengakses file yang terletak di host lokal. Protokol yang digunakan adalah protokol mount, yang mendefinisikan host jarak jauh dan jenis sistem file yang akan digunakan dan dimasukkan ke dalam direktori. Protokol NFS melakukan I/O pada sistem file jarak jauh. Protokol antarmuka dan protokol NFS bekerja dengan RPC dan menyertakan protokol TCP dan UDP. Menggunakan NFS dalam komputasi paralel adalah untuk berbagi data sehingga setiap node slave dapat menjalankan program yang sama pada node master.

1.4. Analisa

Parameter yang diukur dalam perhitungan paralel ini adalah kecepatan dan rasio c/c. Menghitung kecepatan memungkinkan pengguna menentukan efisiensi penggunaan prosesor, dan menghitung rasio c/c memungkinkan pengguna menentukan akurasi. Kecepatan adalah perbandingan kecepatan eksekusi program yang berjalan di komputer paralel versus kecepatan eksekusi program di komputer sekuensial. Efisiensi penggunaan prosesor: Untuk mencapai efisiensi yang tinggi, usahakan agar semua prosesor dalam sistem digunakan secara optimal, tidak ada prosesor yang menganggur sementara prosesor lainnya sedang sibuk (Purbasari, 2005).

Rasio komputasi terhadap komunikasi adalah perbandingan waktu komputasi terhadap waktu komunikasi pada komputer berbasis MIMD berbasis message passing. Rasio c/c merupakan salah satu cara memprediksi efisiensi komputer paralel (Purbasari, 2005).

$$\text{rasio c/c} = t_{\text{comp_par}}/t_{\text{comu}}$$

Granularity adalah jumlah operasi sebelum sinkronisasi. Berdasarkan perincian, algoritma paralel diklasifikasikan menjadi tiga kelompok, yaitu berbutir halus (lebih banyak transfer data daripada perhitungan), berbutir sedang (perhitungan dan transmisi data seimbang), dan berbutir kasar (lebih banyak perhitungan daripada komunikasi). Ini menunjukkan rasio c/c dan perincian. Jika $0 < \text{ratio} < 1$ maka berbutir kasar (Purbasari, 2005).

1.5. Kompleksitas Waktu dan Ruang

Analisis kinerja (efisiensi) algoritma dapat dilakukan dalam dua fase berbeda, yaitu sebelum dan sesudah implementasi. Analisis apriori didefinisikan sebagai analisis teoritis dari suatu algoritma. Performa algoritma diukur dengan mengasumsikan bahwa semua faktor lain, seperti B. kecepatan prosesor, konstan dan tidak berpengaruh pada implementasi. Back Analysis (Analisis posterior) didefinisikan sebagai analisis empiris dari suatu algoritma. Algoritma yang dipilih diimplementasikan dalam bahasa pemrograman. Selain itu, algoritma yang dipilih dijalankan pada komputer target. Analisis ini mengumpulkan data statistik nyata seperti waktu penggunaan dan ruang penyimpanan yang diperlukan. Analisis algoritmik berkaitan dengan eksekusi atau runtime dari berbagai fungsi. Waktu eksekusi suatu operasi dapat didefinisikan sebagai jumlah instruksi komputer yang dieksekusi per operasi (Chakraborty, 2020).

Misalkan X diperlakukan sebagai algoritma dan N diperlakukan sebagai ukuran input data, waktu dan ruang yang diimplementasikan oleh Algoritma X adalah dua faktor utama yang menentukan efisiensi X. Faktor Waktu - Waktu dihitung atau diukur dengan menghitung jumlah operasi kunci seperti perbandingan dalam algoritma pengurutan. Space Factor - Ruang dihitung atau diukur dengan menghitung ruang memori maksimum yang dibutuhkan oleh algoritma. Kompleksitas algoritma $f(N)$ menyediakan waktu berjalan dan/atau ruang penyimpanan yang dibutuhkan oleh algoritma sehubungan dengan N sebagai ukuran data masukan (Chakraborty, 2020).

a. Kompleksitas Ruang

Kompleksitas ruang suatu algoritma merepresentasikan jumlah ruang memori yang dibutuhkan algoritma tersebut dalam siklus hidupnya. Ruang yang dibutuhkan oleh suatu algoritma sama dengan jumlah dari dua komponen berikut: (1) Bagian tetap yang merupakan ruang yang diperlukan untuk menyimpan data dan variabel tertentu (yaitu variabel dan konstanta sederhana, ukuran program, dll.), yang tidak bergantung pada ukuran masalahnya. (2) Bagian variabel adalah ruang yang dibutuhkan oleh variabel, yang ukurannya sangat bergantung pada ukuran masalah. Misalnya, ruang tumpukan rekursi, alokasi memori dinamis, dll. Kompleksitas ruang $S(p)$ dari sembarang algoritma p adalah $S(p) = A + Sp(I)$ di mana A diperlakukan sebagai bagian tetap dan $S(I)$ diperlakukan sebagai bagian variabel dari algoritma yang bergantung pada karakteristik instans I (Chakraborty, 2020)..

Berikut adalah contoh sederhana yang mencoba menjelaskan konsep tersebut

Algoritma SUM(P, Q) Step 1 - START Step 2 - R ← P + Q + 10 Step 3 - Stop
--

Dari algoritma, kita memiliki tiga variabel P, Q dan R dan satu konstanta. Oleh karena itu $S(p) = 1+3$. Sekarang ruang bergantung pada tipe data dari tipe dan variabel konstanta yang diberikan dan akan dikalikan sesuai dengan itu.

b. Kompleksitas Waktu

Kompleksitas waktu dari suatu algoritma adalah representasi dari jumlah waktu yang diperlukan oleh algoritma untuk mengeksekusi hingga selesai. Persyaratan waktu dapat dinyatakan atau didefinisikan sebagai fungsi numerik $t(N)$, di mana $t(N)$ dapat diukur sebagai jumlah langkah, asalkan setiap langkah memerlukan waktu konstan. Misalnya, dalam kasus penambahan dua bilangan bulat n-bit, N langkah diambil. Konsekuensinya, total waktu komputasi adalah $t(N) = c*n$, di mana c adalah waktu yang digunakan untuk penambahan dua bit. Di sini, diamati bahwa $t(N)$ tumbuh secara linear dengan bertambahnya ukuran input (Chakraborty, 2020).

1.6. Speedup Paralel

Percepatan algoritma paralel terhadap algoritma sekuensial yang sesuai adalah rasio waktu komputasi untuk algoritma sekuensial dengan waktu untuk algoritma paralel. Jika faktor percepatan adalah n , maka kita katakan kita memiliki percepatan n -kali lipat. Misalnya, jika algoritma sekuensial membutuhkan waktu komputasi 10 menit dan algoritma paralel yang sesuai membutuhkan 2 menit, kita katakan bahwa ada percepatan 5 kali lipat (Selkie, (n,d)).

Percepatan yang diamati bergantung pada semua faktor implementasi. Misalnya, lebih banyak prosesor sering kali menghasilkan lebih banyak kecepatan; juga, jika program lain berjalan pada prosesor pada saat yang sama dengan program yang mengimplementasikan algoritma paralel, program lain tersebut dapat mengurangi percepatan. Bahkan jika suatu masalah sangat paralel, seseorang jarang benar-benar mendapatkan percepatan n -lipat saat menggunakan prosesor n -lipat. Ada beberapa penjelasan untuk kejadian ini (Selkie, (n,d)).

- Seringkali ada overhead yang terlibat dalam perhitungan. Misalnya, dalam komputasi tata surya, hasilnya harus disalin ke seluruh jaringan pada setiap iterasi. Komunikasi ini penting untuk algoritma, namun waktu yang dihabiskan untuk komunikasi ini tidak secara langsung menghitung lebih banyak solusi untuk masalah n -body. Secara umum, biaya komunikasi sering menjadi kontributor overhead. Waktu pemrosesan untuk menjadwalkan dan mengirimkan proses juga menyebabkan overhead.
- Trices terjadi ketika suatu proses harus menunggu proses lain untuk mengirimkan sumber daya komputasi. Misalnya, setelah setiap komputer di perhitungan tata surya memberikan hasil iterasinya, ia harus menunggu untuk menerima nilai yang diperbarui untuk planet lain sebelum memulai iterasi berikutnya.
- Beberapa bagian dari perhitungan mungkin secara inheren berurutan. Dalam contoh es kutub, hanya perhitungan matriks yang diparalelkan, dan bagian lain memperoleh kinerja hanya karena dilakukan pada perangkat keras dan perangkat lunak yang lebih cepat (pada satu komputer)

Pada kesempatan langka, menggunakan prosesor n dapat menyebabkan lebih dari kecepatan n -kali lipat. Misalnya, jika perhitungan melibatkan kumpulan data besar yang tidak cocok dengan memori utama satu komputer, tetapi cocok dengan memori utama kolektif n komputer, dan jika implementasi paralel yang memalukan hanya membutuhkan bagian proporsional dari data, maka perhitungan paralel yang melibatkan n komputer dapat berjalan lebih dari n kali lebih cepat karena akses disk dapat digantikan oleh akses memori utama (Selkie, (n,d)).

Mengganti akses memori utama dengan akses cache dapat memiliki efek yang serupa. Juga, pemangkasan paralel dalam algoritma backtracking dapat memungkinkan satu proses untuk menghindari perhitungan yang tidak perlu karena pekerjaan sebelumnya dari proses lain (Selkie, (n,d)).

1.7. Hukum Amdahl

Hukum Amdahl adalah rumus untuk memperkirakan percepatan maksimum dari suatu algoritma yang merupakan bagian sekuensial dan bagian paralel. Pencarian untuk bilangan prima 2^k -digit mengilustrasikan masalah seperti ini: Pertama, kita membuat daftar semua k

-digit bilangan prima, menggunakan strategi ayakan berurutan; lalu kita periksa 2k -digit bilangan acak secara paralel sampai kita menemukan bilangan prima (Selkie, (n,d)).

Rumus Hukum Amdahl adalah

$$\text{overall speedup} = \frac{1}{(1 - P) + \frac{P}{S}}$$

- P adalah proporsi waktu dari algoritma yang dapat diparalelkan.
- S adalah faktor percepatan untuk bagian algoritma tersebut karena paralelisasi.

Sebagai contoh, misalkan kita menggunakan strategi kita untuk mencari bilangan prima menggunakan 4 prosesor, dan bahwa 90% waktu berjalan dihabiskan untuk memeriksa bilangan acak 2k digit untuk keutamaan (setelah 10% awal waktu berjalan menghitung daftar k -angka prima). Kemudian $P = .90$ dan $S = 4$ (untuk percepatan 4 kali lipat). Menurut Hukum Amdahl,

$$\text{overall speedup} = \frac{1}{(1 - 0.90) + \frac{0.90}{4}} = \frac{0.10}{0.225} = 3.077$$

Ini memperkirakan bahwa kita akan mendapatkan percepatan 3 kali lipat dengan menggunakan paralelisme 4 kali lipat (Selkie, (n,d)).

Catatan

- Hukum Amdahl menghitung percepatan keseluruhan, dengan mempertimbangkan bahwa bagian berurutan dari algoritma tidak memiliki percepatan, tetapi bagian paralel dari algoritma memiliki percepatan S (Selkie, (n,d))
- Tampaknya mengejutkan bahwa kami hanya mendapatkan percepatan keseluruhan 3 kali lipat ketika 90% algoritma mencapai percepatan 4 kali lipat. Ini adalah pelajaran dari Hukum Amdahl: bagian algoritma yang tidak dapat diparalelkan memiliki efek yang tidak proporsional pada percepatan keseluruhan (Selkie, (n,d)).
- Contoh non-komputasi dapat membantu menjelaskan efek ini. Misalkan sebuah tim yang terdiri dari empat siswa membuat laporan, bersama dengan ringkasan eksekutif, di mana bagian utama laporan membutuhkan 8 jam untuk menulis, dan ringkasan eksekutif memerlukan satu jam untuk menulis dan harus memiliki penulis tunggal (mewakili penulis berurutan). tugas). Jika hanya satu orang yang menulis seluruh laporan, itu akan membutuhkan 9 jam. Tetapi jika empat siswa masing-masing menulis 1/4 dari isi laporan (2 jam, dalam paralelisme 4 kali lipat), kemudian satu siswa menulis ringkasannya, maka waktu yang dihabiskan akan menjadi 3 jam—untuk percepatan keseluruhan 3 kali lipat. . Porsi tugas berurutan memiliki efek yang tidak proporsional karena tiga siswa lainnya tidak melakukan apa-apa selama porsi tugas tersebut (Selkie, (n,d))

Perhitungan singkat menunjukkan mengapa Hukum Amdahl benar (Selkie, (n,d))

1. Misalkan T_s waktu komputasi tanpa paralelisme, dan T_p waktu komputasi dengan paralelisme. Kemudian, percepatan karena paralelisme adalah

$$\text{total speedup} = \frac{T_s}{T_p}$$

2. Nilai P dalam Hukum Amdahl adalah perbandingan T_s yang dapat diparalelkan, suatu bilangan antara 0 dan 1. Maka perbandingan T_s yang tidak dapat diparalelkan adalah 1-P.
3. Ini berarti bahwa

$$T_p = \text{time spent in unparallelizable code} + \text{time spent in parallelizable code} = (1 - P) \times T_s + P \times \frac{T_s}{S}$$

4. Dapat disimpulkan bahwa

$$\text{total speedup} = \frac{T_s}{T_p} = \frac{T_s}{(1 - P) \times T_s + P \times \frac{T_s}{S}} = \frac{1}{(1 - P) + \frac{P}{S}}$$

1.8. Contoh Kasus Analisis Algoritma Paralel

Wibawa (2018) melakukan penelitian tentang komputasi paralel menggunakan model message passing pada SIMRS (Sistem Informasi Manajemen Rumah Sakit). Dengan menggunakan MPI, diharapkan kemampuan komputasi paralel meningkat sehingga mampu menjadi solusi bagi permasalahan beban data dan kecepatan waktu komputasi yang terjadi pada SIMRS. Penelitian dilakukan dengan berfokus pada proses pencarian data manajemen rumah sakit.

Data SIMRS diolah menggunakan program paralel MPI dengan topologi jaringan paralel yaitu menggunakan 1 komputer/ CPU berfungsi sebagai master dan 6 komputer/CPU sebagai slave dengan total 7 komputer/CPU baik master ataupun slave dapat berfungsi mengolah data.

Dari penelitiannya, diperoleh waktu yang dibutuhkan dalam proses eksekusi pada topologi jaringan:

- a. sekvensial = 2,067532 detik
- b. paralel dengan 2 CPU = 1.163986 detik
- c. paralel dengan 3 CPU = 1.075855 detik
- d. paralel dengan 4 CPU = 1.098166 detik
- e. paralel dengan 5 CPU = 1.139087 detik
- f. paralel dengan 6 CPU = 1.163986 detik
- g. paralel dengan 7 CPU = 1.224581 detik

Wibawa (2018) memperhitungkan nilai speed up dengan cara waktu hasil pengolahan data topologi jaringan sekvensial (2, 067532 detik) dibagi waktu pengolahan data menggunakan program paralel MPI sesuai dengan jumlah komputer/CPU yang digunakan. Berikut Tabel nilai *speed up* yang diperoleh

Jumlah komputer/CPU (p)	Waktu paralel (T_p) (detik)	<i>Speed Up (S)</i>
2	1,163986	1,78
3	1,075855	1,92
4	1,098166	1,88
5	1,139087	1,81
6	1,162756	1,78
7	1,224581	1,69

Tabel tersebut menunjukkan adanya peningkatan kecepatan sampai pada penggunaan komputasi paralel pada 3 CPU, namun pada pengujian menggunakan 4,5,6, dan 7 CPU, speed up sudah mulai menurun. Penurunan kecepatan sangat mungkin terjadi berdasarkan Hukum Amdahl yang tercantum dalam buku Kurniawan, A (2010) tentang Pemrograman Paralel MPI & C, bahwa semakin banyak prosesor maka kecepatan akan sampai pada titik jenuh. Secara keseluruhan pengujian speedup pada penelitian Wibawa (2018) jika dibandingkan dengan hasil pengolahan data menggunakan topologi jaringan sekuensial, pengolahan data menggunakan topologi jaringan paralel menggunakan 7 CPU terbukti memiliki kecepatan yang lebih baik.

Setelah itu, dilakukan pula perhitungan efisiensi yang merupakan perbandingan antara speed up dan jumlah CPU yang digunakan. Berikut tabel perhitungan efisiensi.

Jumlah komputer/PU (p)	Waktu paralel (T_p) (detik)	<i>Speed Up (S)</i>	Efisiensi (E)
2	1,163986	1,78	0,89
3	1,075855	1,92	0,64
4	1,098166	1,88	0,47
5	1,139087	1,81	0,36
6	1,162756	1,78	0,30
7	1,224581	1,69	0,24

Penurunan nilai efisiensi yang terjadi pada setiap peningkatan jumlah CPU disebabkan karena adanya CPU yang tidak bekerja (idle). Proses komunikasi yang kompleks menyebabkan terjadinya kondisi dimana CPU tidak bekerja dan menunggu untuk mendapat perintah. Hasil nilai efisiensi menunjukkan semakin banyak penggunaan CPU maka semakin menurun nilai efisiensi.

2. TEKNIK PERANCANGAN ALGORITMA PARALEL

2.1. Teknik Desain

Memilih teknik desain yang tepat untuk algoritma paralel adalah tugas yang paling sulit dan penting. Sebagian besar masalah pemrograman paralel dapat memiliki lebih dari satu solusi. Dalam bab ini kita membahas teknik desain berikut untuk algoritma paralel:

a. Divide and conquer

Dalam pendekatan Divide and Conquer, masalah dibagi menjadi submasalah kecil. Kemudian sub masalah diselesaikan secara rekursif dan digabungkan untuk menyelesaikan masalah aslinya. Pendekatan bagi dan taklukkan melibatkan langkah-langkah berikut di setiap level: (Tutorialspoint. (n.d)).

- 1) Split - Masalah asli dibagi menjadi beberapa sub masalah.
- 2) Conquer - Sub Masalah diselesaikan secara rekursif.
- 3) Menggabungkan - Solusi untuk submasalah digabungkan, menghasilkan solusi untuk masalah awal (Tutorialspoint. (n.d)).

Pendekatan bagi dan taklukkan diimplementasikan dalam algoritma: binary search, quick sort, merge sort, integer multiplication, matrix inversion, matrix multiplication.

b. Greedy Method

Algoritma pengoptimalan solusi serakah memilih solusi terbaik setiap saat. Algoritma Greedy sangat mudah diterapkan pada masalah yang kompleks. Ini memutuskan langkah mana di langkah selanjutnya yang akan memberikan solusi paling akurat. Algoritma ini disebut serakah karena mengabaikan seluruh program secara keseluruhan ketika memberikan solusi optimal untuk kasus yang lebih kecil. Setelah solusi dipertimbangkan, algoritma serakah tidak pernah mempertimbangkan solusi yang sama lagi. algoritma Greedy bekerja secara rekursif dan membuat grup objek dari komponen sekecil mungkin. Rekursi adalah proses pemecahan masalah di mana pemecahan masalah yang diberikan bergantung pada penyelesaian contoh masalah yang lebih kecil (Tutorialspoint. (n.d)).

c. Dynamic Programming

Pemrograman dinamis adalah teknik optimisasi yang membagi masalah menjadi submasalah yang lebih kecil dan, setelah menyelesaikan setiap submasalah, menggabungkan semua solusi untuk mendapatkan solusi akhir. Tidak seperti membagi dan menaklukkan, pemrograman dinamis berulang kali menggunakan kembali solusi untuk submasalah (Tutorialspoint. (n.d)). Contoh pemrograman dinamis adalah algoritma rekursif urutan Fibonacci.

d. Backtracking Algoritma

Backtracking adalah salah satu teknik optimasi untuk menyelesaikan masalah kombinatorial. Ini diterapkan untuk masalah pemrograman dan dunia nyata. Masalah delapan ratu, teka-teki Sudoku, dan melintasi labirin adalah contoh populer yang menggunakan algoritma mundur. Dalam retret kita mulai dari solusi yang memungkinkan yang memenuhi semua kondisi yang diperlukan. Kemudian kita pindah ke level berikutnya dan jika tidak ada solusi yang memuaskan kita kembali ke satu level dan memulai alternatif baru (Tutorialspoint. (n.d)).

e. Branch and Bound

Algoritma branch and bound adalah teknik optimasi yang memberikan solusi optimal untuk suatu masalah. Itu mencari seluruh ruang solusi untuk solusi terbaik untuk masalah yang diberikan. Kondisi batas dari fungsi yang akan dioptimalkan digabungkan

dengan nilai saat ini dari solusi terbaik. Ini memungkinkan algoritma untuk menemukan bagian dari seluruh ruang solusi (Tutorialspoint. (n.d)).

Tujuan pencarian Branch and Bound adalah untuk mempertahankan jalur biaya terendah ke tujuan. Setelah solusi ditemukan, solusi tersebut dapat ditingkatkan lebih lanjut. Pencarian cabang dan terikat diimplementasikan dalam pencarian terbatas kedalaman dan pencarian mendalam pertama (Tutorialspoint. (n.d)).

f. pemrograman linier

Pemrograman linier menggambarkan kelas tugas optimisasi yang luas di mana kriteria dan batasan optimisasi adalah fungsi linier. Ini adalah teknik untuk mendapatkan hasil terbaik, seperti B. keuntungan tertinggi, jalur terpendek atau biaya terendah. Dalam pemrograman ini, kita harus memiliki sekumpulan variabel dan harus menetapkannya nilai absolut untuk memenuhi sekumpulan persamaan linier dan memaksimalkan atau meminimalkan fungsi linier objektif yang diberikan (Tutorialspoint. (n.d)).

2.2. Compiler Design

a. Diagram Dependensi

Diagram dependensi digunakan untuk menunjukkan aliran informasi antar atribut dalam parse tree. Di pohon parse, grafik dependensi pada dasarnya membantu menentukan urutan evaluasi atribut. Tujuan utama dari grafik ketergantungan adalah untuk membantu kompiler memeriksa berbagai jenis ketergantungan antara ekspresi sehingga mereka tidak berakhir berantakan, yaitu. H. dengan cara yang mempengaruhi arti program. Ini adalah aspek kunci yang membantu mengidentifikasi berbagai komponen program yang dapat diparalelkan (Geeksforgeeks. (n.d)).

Hal ini membantu untuk menentukan dampak dari perubahan dan objek yang terpengaruh. Tapi gambar untuk menghubungkan tindakan dependen dapat digunakan untuk mendeskripsikan dependensi. Busur ini menghasilkan pengurutan sebagian antara operasi dan juga mencegah program berjalan secara paralel. Meskipun rantai definisi pengguna adalah jenis analisis ketergantungan, ini mengarah pada penilaian ketergantungan data yang terlalu hati-hati. Ada empat jenis ketergantungan antara pernyataan I dan j di jalur kontrol umum (Geeksforgeeks. (n.d)).

Grafik dependensi, seperti jaringan terarah lainnya, memiliki simpul atau simpul yang direpresentasikan sebagai kotak atau lingkaran dengan nama dan panah yang menghubungkannya ke arah jalur yang diperlukan. Diagram ketergantungan banyak digunakan dalam literatur ilmiah untuk menggambarkan hubungan semantik, ketergantungan temporal dan kausal antara peristiwa, dan aliran arus listrik dalam rangkaian elektronik. Menggambar diagram ketergantungan sangat umum dalam ilmu komputer sehingga kami ingin menggunakan alat yang akan mengotomatiskan proses berdasarkan beberapa instruksi teks dasar yang kami berikan (Geeksforgeeks. (n.d)).

Dependensi secara kasar dibagi ke dalam beberapa kategori, seperti berikut:

1) Ketergantungan data (Data Dependencies)

Saat ekspresi menghitung data yang digunakan ekspresi lain. Keadaan di mana pernyataan harus menunggu hasil dari pernyataan sebelumnya sebelum dapat

menyelesaikan eksekusinya. Ketergantungan data memicu gangguan dalam aliran layanan pipelined dari prosesor atau mencegah paralelisasi instruksi pada prosesor superscalar pada prosesor berkinerja tinggi menggunakan metode pipeline atau superscalar (Geeksforgeeks. (n.d)).

2) Manajemen Kecanduan (Control Dependencies)

Ketergantungan kontrol adalah yang berasal dari aliran kontrol program yang tepat. Skenario dimana instruksi program dieksekusi ketika instruksi sebelumnya telah dievaluasi sehingga dapat dieksekusi disebut ketergantungan kontrol (Geeksforgeeks. (n.d)).

3) Ketergantungan aliran (Flow Dependencies)

Dalam ilmu komputer, ketergantungan aliran terjadi ketika pernyataan program mereferensikan data dari pernyataan sebelumnya (Geeksforgeeks. (n.d)).

4) Melawan Kecanduan (Anti Dependencies)

Ketika sebuah instruksi membutuhkan nilai yang kemudian diubah, ini dikenal sebagai anti-ketergantungan atau tulisan WAR. Pernyataan 2 mencegah ketergantungan pernyataan 3 dalam contoh berikut; Serangkaian pernyataan ini tidak dapat dimodifikasi dan dieksekusi secara paralel (mungkin mengubah urutan pernyataan) karena ini mengubah nilai akhir dari A (Geeksforgeeks. (n.d)).

5) Ketergantungan Keluaran (Output Dependency)

Ketergantungan output, juga dikenal sebagai write-by-write (WAW), terjadi ketika urutan eksekusi instruksi mempengaruhi nilai output akhir dari suatu variabel. Dalam contoh berikut, terdapat dependensi keluaran antara pernyataan 3 dan 1; Mengubah urutan pernyataan mempengaruhi nilai akhir A, sehingga pernyataan ini tidak dapat dijalankan secara paralel (Geeksforgeeks. (n.d)).

6) Manajemen Kecanduan (Control-Dependency)

Jika hasil A menentukan apakah B harus dijalankan atau tidak, pernyataan B memiliki ketergantungan kontrol pada pernyataan A sebelumnya. Gaya tampilan perintah S 2S 2 memiliki ketergantungan kontrol pada gaya tampilan perintah S 1S 1 berikut ini contoh. Namun, gaya tampilan S 3S 3 tidak bergantung pada gaya tampilan S 1S 1 karena gaya tampilan S 3S 3 selalu dijalankan terlepas dari hasil gaya tampilan S 1S 1 (Geeksforgeeks. (n.d)).

b. Granularity

Granularity adalah ukuran komponen, atau deskripsi dokumen, yang membentuk suatu sistem. Granularity adalah ukuran relatif, skala, tingkat detail atau kedalaman penetrasi yang mencirikan suatu objek atau aktivitas. Dengan kata lain, granularity ukuran sejauh mana entitas yang lebih besar dibagi lagi. Misalnya, satu *yard* yang dipecah menjadi beberapa inci memiliki perincian yang lebih halus daripada satu *yard* yang dipecah menjadi kaki. (chemeuropa.com)

Dalam pemrosesan komputer paralel, granularity mengacu pada jumlah perhitungan yang dilakukan secara paralel relatif terhadap ukuran keseluruhan program, yaitu ukuran kualitatif dari rasio komputasi terhadap komunikasi.

Berdasarkan jumlah pekerjaan yang dilakukan oleh tugas paralel, parallelisme dapat diklasifikasikan menjadi tiga kategori: (alchetron.com)

1) Butir kasar (*Fine-grained*)

Dalam sistem berbutir kasar, bagian paralel relatif besar dan itu berarti lebih banyak komputasi dan lebih sedikit komunikasi. Sejumlah besar perhitungan dan pekerjaan dilakukan. Butiran kasar memiliki perhitungan tinggi untuk proporsi korespondensi dan menyarankan kebebasan yang lebih besar untuk peningkatan eksekusi

2) Butir halus (*Coarse-grained*)

Dalam sistem butiran halus, bagian paralel relatif kecil dan itu berarti komunikasi lebih sering dilakukan. Rasio komputasi terhadap komputasi rendah dan membutuhkan overhead komunikasi yang tinggi. Jika bagian paralel terlalu kecil maka overhead yang diperlukan untuk komunikasi dan sinkronisasi antar tugas bisa membutuhkan waktu lebih lama daripada komputasi.

3) Butir sedang (*Medium-grained*)

Sistem butiran sedang adalah kompromi antara sistem berbutir kasar dan sistem berbutir halus., dimana kita memiliki ukuran tugas dan waktu komunikasi yang lebih besar daripada sistem berbutir halus dan lebih rendah daripada sistem berbutir kasar

Tingkat granularity dalam suatu sistem ditentukan oleh algoritma yang ditetapkan dan lingkungan perangkat keras tempatnya berjalan. Komunikasi data yang diperlukan untuk memulai proses besar mungkin membutuhkan banyak waktu. Di sisi lain, proses besar seringkali memiliki lebih sedikit komunikasi yang harus dilakukan selama pemrosesan. Suatu proses mungkin hanya memerlukan sejumlah kecil informasi untuk memulai pemrosesan, tetapi mungkin perlu mendapatkan lebih banyak informasi/ data untuk melanjutkan pemrosesan, atau mungkin perlu melakukan banyak komunikasi dengan proses lain untuk melakukan pemrosesannya.

c. Concurrency

Konkurensi adalah eksekusi dari beberapa urutan instruksi pada waktu yang sama. Itu terjadi di sistem operasi ketika ada beberapa tugas yang berjalan secara paralel. Tugas proses yang berjalan selalu berkomunikasi satu sama lain melalui *shared memory* atau *passing message*. Hasil konkurensi dalam berbagi sumber daya mengakibatkan masalah seperti kebuntuan dan kekurang sumber daya. Ini membantu dalam teknik seperti mengoordinasikan eksekusi proses, alokasi memori, dan penjadwalan eksekusi untuk memaksimalkan *throughput*.

Prinsip konkurensi, yaitu 1) baik proses yang disisipkan maupun yang tumpang tindih dapat dilihat sebagai contoh dari proses yang bersamaan, keduanya menghadirkan masalah yang sama; 2) kecepatan relatif eksekusi tidak dapat diprediksi, tergantung pada kegiatan proses lainnya, cara sistem operasi menangani interupsi, dan kebijakan penjadwalan sistem operasi.

Dalam konkurensi, ditemukan berbagai masalah di dalamnya, yaitu 1) sulit berbagi sumber daya global dengan aman; 2) sulit bagi sistem operasi untuk mengelola alokasi sumber daya secara optimal; 3) sangat sulit menemukan kesalahan pemrograman karena laporan biasanya tidak dapat direproduksi; 4) mungkin tidak efisien bagi sistem operasi untuk hanya mengunci saluran dan mencegah penggunaannya oleh proses lain.

Keuntungan konkurensi, yaitu: 1) memungkinkan untuk menjalankan banyak aplikasi secara bersamaan; 2) pemanfaatkan sumber daya yang lebih baik karena memungkinkan sumber daya yang tidak digunakan oleh satu aplikasi dapat digunakan untuk aplikasi lain; 3) waktu respons rata-rata yang lebih baik karena proses tidak harus berjalan secara berurutan; 4) performa yang lebih baik karena ketika satu aplikasi hanya menggunakan prosesor dan aplikasi lain hanya menggunakan *disk drive* maka waktu untuk menjalankan dua aplikasi secara bersamaan hingga selesai akan lebih singkat dibandingkan dengan waktu untuk menjalankan setiap aplikasi secara berurutan.

Disamping kelebihan, konkurensi juga memiliki kelemahan, yaitu: 1) diperlukan sistem untuk melindungi beberapa aplikasi; 2) diperlukan sistem untuk mengkoordinasi banyak aplikasi melalui mekanisme tambahan; 3) overhead kinerja tambahan dan kompleksitas dalam sistem operasi diperlukan untuk beralih di antara aplikasi.; 4) terkadang menjalankan terlalu banyak aplikasi secara bersamaan menyebabkan penurunan kinerja yang parah.

Terdapat juga isu/ topik penting terkait konkurensi, yaitu: 1) operasi yang non-atomik tetapi dapat disela oleh banyak proses dapat menyebabkan masalah; 2) kondisi balapan terjadi pada hasil tergantung pada proses mana yang mencapai titik pertama; 3) proses dapat memblokir menunggu sumber daya; 4) *starvation*, terjadi ketika proses tidak mendapatkan layanan untuk *progress* 5) kebuntuan, terjadi ketika dua proses diblokir dan karenanya tidak dapat melanjutkan untuk dieksekusi.

d. Critical Path

Critical path adalah metode yang digunakan dalam perencanaan *project*, umumnya untuk penjadwalan *project* agar penyelesaiannya tepat waktu sehingga dapat membantu dalam penentuan waktu paling awal dimana seluruh *project* dapat diselesaikan. Ada dua konsep utama dalam metode ini, yaitu *critical task* dan *critical path*. *Critical task* adalah tugas proses yang tidak dapat ditunda karena penyelesaian keseluruhan *project* akan tertunda maka harus diselesaikan tepat waktu sebelum memulai proses yang lain. *Critical path* adalah urutan proses dan merupakan jalur terbesar dalam jaringan proses sehingga memberi waktu minimum yang diperlukan untuk menyelesaikan seluruh *project*. Kegiatan di *critical path* dikenal sebagai kegiatan kritis dan jika kegiatan ini tertunda maka penyelesaian keseluruhan *project* juga tertunda.

e. Task Interaction

Tugas secara umum menukar data dengan yang lain dalam sebuah dekomposisi. Grafik tugas (node) dan pertukaran interaksi/data disebut sebagai grafik interaksi tugas. Grafik interaksi tugas mewakili ketergantungan data, sedangkan grafik ketergantungan tugas mewakili ketergantungan kontrol. Secara umum, jika granularity dari dekomposisi adalah halus, *overhead* terkait (sebagai rasio pekerjaan yang bermanfaat terkait dengan tugas) meningkat. (Gram)

f. Processes and Mapping

Secara umum, jumlah tugas dalam dekomposisi melebihi jumlah elemen pemrosesan yang tersedia. Untuk alasan ini, algoritma paralel juga harus menyediakan sebuah pemetaan

tugas ke proses. Pemetaan tugas yang tepat untuk proses sangat penting untuk kinerja paralel dari suatu algoritma. Pemetaan ditentukan oleh ketergantungan tugas dan grafik interaksi tugas. Grafik ketergantungan tugas dapat digunakan untuk memastikan pekerjaan itu tersebar merata di semua proses pada titik mana pun (minimum *idling* dan keseimbangan beban optimal). Grafik interaksi tugas dapat digunakan untuk memastikan proses membutuhkan interaksi minimum dengan proses lain (minimal komunikasi). Pemetaan yang tepat harus meminimalkan waktu eksekusi paralel oleh: 1) pemetaan tugas independen untuk proses yang berbeda; 2) menugaskan tugas pada jalur kritis ke proses segera setelah mereka menjadi *available*; 3) meminimalkan antar proses dengan memetakan tugas dengan interaksi padat untuk proses yang sama.

Kesimpulan

1. Dalam ilmu komputer, algoritma paralel, berbeda dengan algoritma sekuensial (atau serial) tradisional, adalah algoritma yang dapat dieksekusi secara berurutan oleh beberapa proses berbeda dan kemudian digabungkan kembali menjadi satu. Pengguna akan mendapatkan hasil yang tepat.
2. Algoritma paralel sendiri memiliki beberapa teknik perancangan maupun analisis kompleksitas.
3. Teknik perancangan algoritma paralel sendiri meliputi desain teknik dan desain *compiler*, yang masing-masing memiliki kategorinya sendiri.

Daftar Pustaka

Alchetron (n.d). *Granularity (Parallel Computing)*. diakses pada tanggal 23 November 2022 dari [https://alchetron.com/Granularity-\(parallel-computing\)](https://alchetron.com/Granularity-(parallel-computing))

Chemeeurope (n.d.) *Granularity*. diakses tanggal 23 November 2022 di https://www.chemeurope.com/en/encyclopedia/Granularity.html#_note-0/

Dadox, angga. 2013. Analisis Algoritma Paralel. diakses pada tanggal 19 November 2022 dari https://www.slideshare.net/angga_dadox/analisa-algoritma-paralel.

Gama et al. _____. *Chapter 3 Principles of Parallel Algorithm Design*. diakses tanggal 24 November 2022 di https://www.uio.no/studier/emner/matnat/ifi/INF3380/v13/undervisningsmateriale/chap3_selected_slides.pdf

Geeksforgeeks (n.d.) *Concurrency in Operating System..* diakses tanggal 23 November 2022 dari <https://www.geeksforgeeks.org/concurrency-in-operating-system/>

Geeksforgeeks. (n.d). Dependency Graph in Compiler Design. diakses pada tanggal 19 November 2022 di <https://www.geeksforgeeks.org/dependency-graph-in-compiler-design/>

Purbasari, Ayu. 2005. Teknis dan Strategi Pembangunan Algoritma Paralel. diakses pada tanggal 19 November 2022 dari <https://pbasari.files.wordpress.com/2014/12/teknisdanstrategipemrogramanparalel.pdf>

Chakraborty, Anab. 2020. Kompleksitas Waktu dan Ruang dalam Struktur Data. diakses pada tanggal 19 November 2022 dari <https://www.tutorialspoint.com/time-and-space-complexity-in-data-structure>

Selkie, Macalester. (n.d). Parallel Speedup. diakses pada tanggal 19 November 2022 di <http://selkie.macalester.edu/csinparallel/modules/IntermediateIntroduction/build/html/ParallelSpeedup/ParallelSpeedup.html>

Tech4Solution (n.d.). *What is Granularity*. diakses tanggal 23 November 2022 di <https://www.tech4solution.com/2021/05/what-is-granularity.html>

Tutorialspoint. (n.d). Paralel Algorithm-Desain Techniques. diakses pada tanggal 19 November 2022 di https://www.tutorialspoint.com/parallel_algorithm/design_techniques.htm

Lampiran

Slide

presentasi:

https://univindonesia-my.sharepoint.com/:p/g/personal/cintya_kusuma_office_ui_ac_id/Ebl1uU8HTmFBkmJzznGAJHgBvriZJpuhzLHkQeHIplbnGA?e=ZLq5td

Video:

Part1: <https://youtu.be/SoBwkIp7C9k>

Part2: <https://youtu.be/lVk0Y-rhhXA>

BAB 6

Implementasi Algoritma Paralel di Beberapa Platform Bahasa Pemrograman

Cintya Kusuma Mahadhika, Elizabeth Lilies Megawati, Eka Dwi Agustina Ginting

Abstrak

Algoritma paralel adalah algoritma yang dapat dieksekusi dalam satu waktu pada banyak perangkat processing yang berbeda. Implementasi algoritma paralel dapat dijalankan pada beberapa program. Dalam makalah ini hanya dijelaskan beberapa program, yaitu *python*, *matlab*, *C*, dan *CUDA*.

Kata kunci: algoritma paralel, algoritma Python, algoritma Matlab, algoritma C, algoritma C++, CUDA

Pendahuluan

Algoritma paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer independent. Hal ini umumnya diperlukan saat kapasitas komputasi yang diperlukan sangat besar, baik karena pengolahan data maupun karena tuntutan proses komputasi yang banyak. Pada umumnya kasus ini ditemui yakni komputasi numerik yaitu menyelesaikan persamaan matematis di bidang fisika, kimia, teknik dan bidang lainnya. Permasalahan-permasalahan pada bidang tersebut diselesaikan dengan komputasi paralel menggunakan berbagai aplikasi yang sesuai. Pada makalah ini akan dijelaskan beberapa aplikasi beserta algoritmanya dalam mengimplementasi algoritma paralel.

Pembahasan

1. Implementasi Algoritma Paralel pada Matlab & Python

a. Implementasi algoritma paralel pada Python

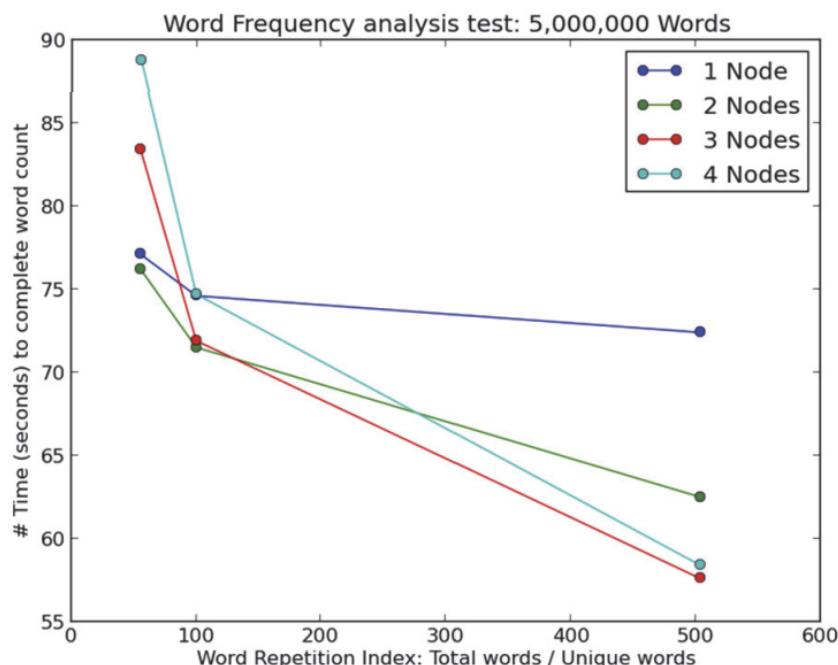
Python merupakan bahasa pemrograman yang memiliki struktur data tingkat tinggi yang efisien, pendekatan yang sederhana namun efektif untuk pemrograman berorientasi objek. Python memiliki beberapa modul sintaks dan paket yang dapat digunakan sesuai kebutuhan. Sintaks Python mempunyai sifat yang dinamis sehingga dinilai baik untuk *scripting* dan berkembang dengan cepat. Python banyak digunakan untuk beberapa masalah seperti pemrosesan array multi dimensi, aljabar linier, 2D dan 3D visualisasi, dan tugas komputasi ilmiah lainnya (Dalcin et. al., 2011)

Lebih lanjut, Dalci (2011) menyatakan bahwa paket maupun modul dalam Python dapat dikembangkan oleh pengguna. Beberapa paket yang acap kali digunakan dalam Python adalah NumPy, SWIG, F2PY, dan Cython. Beberapa penelitian juga telah memaparkan kinerja Python dalam menyelesaikan masalah tertentu.

Salah satu penggunaan Python dalam algoritma parallel adalah proses analisis banyaknya kata yang muncul (Lescisin, 2017). Adapun algoritma untuk kasus tersebut adalah sebagai berikut.

- a. Diasumsikan bahwa file yang akan dianalisis ada di semua node dalam klaster.
- b. Tetapkan file ke setiap node.
- c. Setiap node akan membaca bagian dari file dan membuat tabel hash (kamus Python) yang menunjukkan seberapa sering setiap kata muncul. Kuncinya akan menjadi kata dan frekuensi kemunculan kata tersebut.
- d. Semua tabel hash ini akan dikembalikan ke simpul utama
- e. Node master kemudian akan mendorong tabel hash ini dalam mode dua-dua untuk node di mana mereka akan berada digabungkan dan dikembalikan menjadi satu. Proses ini akan dilanjutkan sampai hanya satu tabel yang mewakili kata dengan frekuensi file tetap.

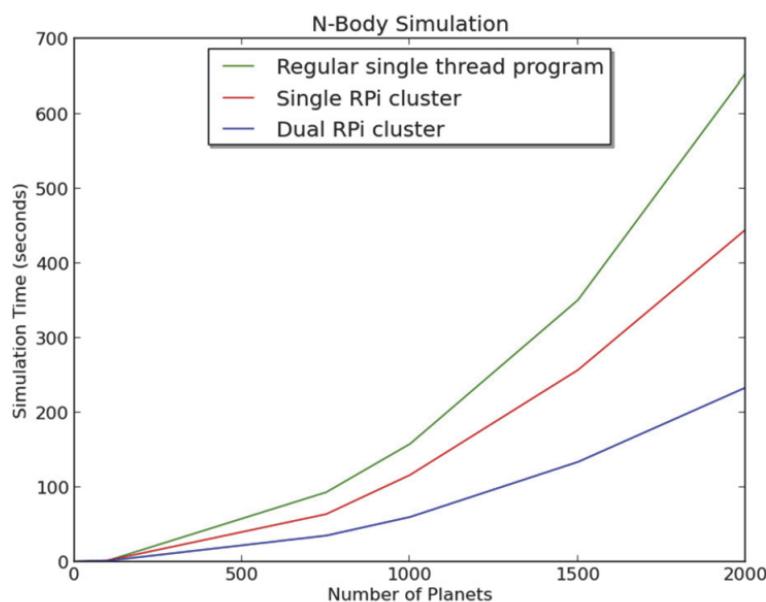
Adapun hasil perhitungan kata dari data sebanyak 5000000 kata adalah sebagai berikut.



Contoh lain dari aplikasi pemrograman paralel dengan Python adalah simulasi *N-body* (Lescisin, 2017). Simulasi ini menggambarkan bagaimana sejumlah objek akan berinteraksi jika terdapat posisi awal, kecepatan awal, dan massa. Algoritma dari simulasi ini adalah sebagai berikut.

1. Daftar planet disediakan sebagai file JSON yang berisi daftar kamus yang mewakili planet benda dengan massa, koordinat x, koordinat y, dan pengenal numerik. Daftar ini dibagikan kepada semua node dalam cluster.
2. Setiap node ditugaskan satu n (di mana n adalah jumlah node dalam cluster) dari daftar yang diperlukan untuk menghitung gaya total pada planet-planet ini ke semua planet dalam daftar.
3. Daftar ini akan memuat himpunan bagian dari set planet lengkap digabungkan bersama-sama di node master yang menghasilkan himpunan lengkap planet di mana gaya masing-masing telah diketahui.

Adapun performa dari simulasi tersebut adalah sebagai berikut.



b. Implementasi algoritma paralel pada Matlab

MATLAB adalah lingkungan komputasi yang biasa digunakan oleh para insinyur, peneliti, dan ekonom. Alasan utama memilih MATLAB untuk komputasi algoritma DIC adalah karena MATLAB memungkinkan pengguna untuk mengimplementasikan dan menguji algoritma yang ditentukan pengguna. Ini memiliki database besar algoritma bawaan yang dapat diwarisi pengguna dan menggunakan berdasarkan persyaratan masalah. Selain basis data bawaan, perpustakaan eksternal juga dapat digunakan dengan mengimportnya ke antarmuka MATLAB. Dimungkinkan juga untuk melakukan analisis dan visualisasi data yang luas menggunakan MATLAB. MATLAB hadir dengan banyak set alat dan fungsi yang kuat yang membuatnya relatif mudah untuk menggunakan fungsionalitas tingkat tinggi dan mengintegrasikan fungsi kompleks ke dalam program yang dikembangkan sendiri (Jiang, 2019).

Salah satu implementasi pemrograman paralel dengan Matlab adalah algoritma *Digital Image Correlation* (DIC). Ide utama DIC adalah untuk menentukan perpindahan antara dua keadaan dari suatu objek yang menarik dengan membandingkan dua gambar dari objek ini di dua keadaan yang berbeda; gambar referensi seperti sampel dalam kondisi tidak terdeformasi dan gambar target seperti sampel dalam kondisi terdeformasi. Untuk tujuan ini terdapat dua fungsi, satu mewakili tingkat keabuan pada gambar referensi $f(x, y)$ dan yang lainnya untuk gambar target $g(x^*, y^*)$ pada posisi $[x, y]$ dan $[x^*, y^*]$, masing-masing didefinisikan. Gambar referensi dibagi menjadi beberapa sub-gambar persegi panjang. Setiap sub-gambar memiliki distribusi tingkat keabuan spesifik yang dibandingkan dengan sub-gambar dengan ukuran yang sama di dalam gambar target. Perbandingan ini dilakukan dengan menggunakan koefisien korelasi, yang memberikan ukuran kesamaan (Thoma et. al., 2021). Penelitian tersebut membandingkan performa beberapa algoritma dari dua komputer yang berbeda untuk membandingkan performa kedua komputer tersebut. Dibandingkan pula pencarian *integer-pixel* dan algoritma terkait.

Integer-Pixel Search	Sub-Pixel Search Algorithm	Time PC 1 [s]		Time PC 2 [s]	
		Set 1	Set 2	Set 1	Set 2
Brute Force	Newton-Raphson	2682.5	58.66	1905.4	36.722
PSO	Newton-Raphson	863.5	20.00	677.6	8.38
Mod. PSO	Newton-Raphson	861.5	19.41	618.9	8.17
Brute Force	IC-GN	2938.0	61.56	1942.1	38.756
PSO	IC-GN	914.6	24.18	678.8	11.10
Mod. PSO	IC-GN	964.6	24.15	685.7	10.99
Brute Force	IC-GN by Baker [35]	2348.4	55.13	1776.4	35.00
PSO	IC-GN by Baker [35]	552.2	17.98	424.2	7.49
Mod. PSO	IC-GN by Baker [35]	902.7	20.19	418.9	6.60

Dari tabel tersebut, terlihat bahwa kecepatan algoritma bergantung pada masalah *integer-pixel* maupun komputer yang digunakan.

2. Implementasi Algoritma Paralel pada Bahasa C

OpenMP (Open MultiProcessing) merupakan sistem bahasa pemrograman paralel yang digunakan untuk mengekspresikan paralelisme *shared-memory*. (Mohr dkk,2001). Lebih lengkapnya, OpenMP adalah *Application Program Interface (API)* untuk menulis aplikasi *multithreaded* (MT), berupa satu set direktif *compiler* dan *library* untuk pemrograman aplikasi paralel yang menyederhanakan penulisan program pada C++, C, Fortran. Multithreading adalah kemampuan CPU untuk menjalankan beberapa proses dalam waktu yang bersamaan, Kemampuan ini dapat mengurangi proses runtime dan meningkatkan efisiensi algoritma. Selain itu, *shared-memory* adalah arsitektur paralel yang memungkinkan setiap proses berjalan untuk mendapatkan sumber memori secara proporsional. Melalui fitur ini, waktu akses ke penyimpanan dapat ber minimum dan kecepatan proses runtime dapat ditingkatkan. (Rabbani, 2018)

Program OpenMP memiliki bagian yang berurutan dan bagian yang paralel. Secara umum program OpenMP dimulai dengan bagian berurutan di mana ia mengatur lingkungan, menginisialisasi variabel, dan sebagainya. Saat dijalankan, program OpenMP akan menggunakan satu utas (di bagian berurutan), dan beberapa utas (di bagian paralel). Terdapat satu utas yang berjalan dari awal hingga akhir disebut *master thread*. Bagian paralel dari program akan menyebabkan utas tambahan bercabang yang disebut *slave threads*. Bagian kode yang akan dieksekusi secara paralel ditandai dengan arahan khusus (omp pragma). Ketika eksekusi mencapai bagian paralel (ditandai oleh omp pragma), arahan ini akan menyebabkan *slave threads* terbentuk. Setiap utas mengeksekusi bagian paralel dari kode secara independen. Saat sebuah utas selesai, utas tersebut bergabung dengan *master*. Ketika semua tas selesai, *master* melanjutkan dengan kode yang mengikuti bagian paralel. Setiap utas memiliki ID yang melekat pada utas tersebut yang dapat diperoleh dengan menggunakan fungsi *library runtime* (disebut `omp_get_thread_num()`). ID dari utas utama adalah 0.(Tim Matson)

OpenMp lebih efisien dan memungkinkan kode paralel tingkat rendah, namun OpenMP menyembunyikan detail tingkat rendah dan memungkinkan pemrogram untuk mendeskripsikan kode paralel dengan konstruksi tingkat tinggi sesederhana mungkin. OpenMP memiliki arahan yang memungkinkan programmer untuk menentukan: daerah paralel, apakah variabel di bagian paralel bersifat pribadi atau bersama, bagaimana jika utas disinkronkan, cara memparalelkan loop, dan bagaimana pekerjaan dibagi antara utas (penjadwalan)

Contoh implementasi algoritma paralel pada Bahasa C

Bastian dkk (2022) melakukan penelitian untuk mengimplementasi pemrograman paralel menggunakan platform OpenMP pada citra digital dengan metode *Low-Pass Filter* dan *histogram equalization*. Tujuan dari penelitian tersebut adalah mempercepat proses pengolahan citra digital.

Citra digital yang akan diproses dikonversikan terlebih dahulu ke dalam matriks dua dimensi dengan menggunakan bantuan aplikasi Octave. Kemudian semua data matriks citra dibaca dan dilakukan 2 tahap pemrosesan (*Low Pass Filter* dan *Histogram Equalization*) menggunakan program bahasa C yang menerapkan teknik pemrograman paralel menggunakan platform OpenMP.

Berikut ini algoritma metode *Low-Pass Filter* untuk mengurangi *noise* berdasarkan persamaan.

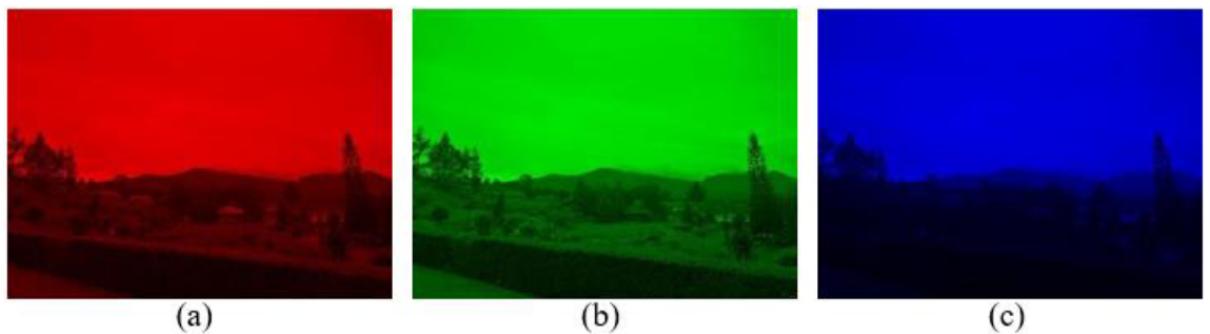
```
#pragma omp parallel
for i=0 to N-1
for j=0 to M-1
if i=0 OR j=0 OR i=N-1 OR j=M-1 then
outputR(i,j) <- inputR(i,j)
outputG(i,j) <- inputG(i,j)
outputB(i,j) <- inputB(i,j)
else
outputR(i,j) <- (1/k) * (W(0,0) *
inputR(i-1,j-1)
+W(0,1) * inputR(i-1,j)
+W(0,2) * inputR(i-1,j+1)
+W(1,0) * inputR(i,j-1)
```

```

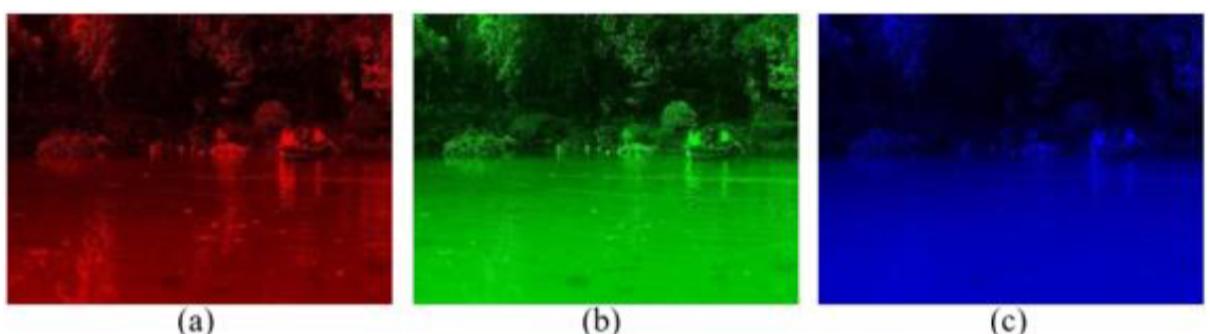
+W(1,1) * inputR(i,j)
+W(1,2) * inputR(i,j+1)
+W(2,0) * inputR(i+1,j-1)
+W(2,1) * inputR(i+1,j)
+W(2,2) * inputR(i+1,j+1)
outputG(i,j) <- (1/k) * (W(0,0) *
inputG(i-1,j-1)
+W(0,1) * inputG(i-1,j)
+W(0,2) * inputG(i-1,j+1)
+W(1,0) * inputG(i,j-1)
+W(1,1) * inputG(i,j)
+W(1,2) * inputG(i,j+1)
+W(2,0) * inputG(i+1,j-1)
+W(2,1) * inputG(i+1,j)
+W(2,2) * inputG(i+1,j+1)
outputB(i,j) <- (1/k) * (W(0,0) *
inputB(i-1,j-1)
+W(0,1) * inputB(i-1,j)
+W(0,2) * inputB(i-1,j+1)
+W(1,0) * inputB(i,j-1)
+W(1,1) * inputB(i,j)
+W(1,2) * inputB(i,j+1)
+W(2,0) * inputB(i+1,j-1)
+W(2,1) * inputB(i+1,j)
+W(2,2) * inputB(i+1,j+1))
endif
endfor
endfor

```

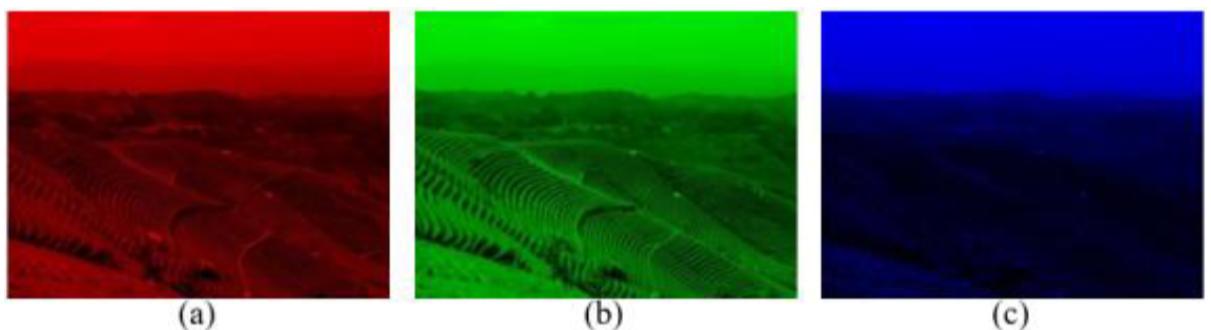
Setelah program dijalankan, diperoleh warna yang dihasilkan dari citra digital berdasarkan kanal warnanya.



Gambar Sampel citra digital pertama setelah dilakukan pemecahan kanal warna



Gambar Sampel citra digital kedua setelah dilakukan pemecahan kanal warna



Gambar Sampel citra digital ketiga setelah dilakukan pemecahan kanal warna

Kemudian dilakukan perbaikan kontras pada matriks citra digital dengan menggunakan metode *Histogram Equalization* berdasarkan persamaan.

```
#pragma omp parallel
for i=0 to N-1
    for j=0 to M-1
        kR <- inputR(i,j)
        kG <- inputG(i,j)
        kB <- inputB(i,j)
        histogramR(kR) <- histogramR(kR) + 1
        histogramG(kG) <- histogramG(kG) + 1
        histogramB(kB) <- histogramB(kB) + 1
    endfor
endfor
#pragma omp parallel
for i=0 to GrayLevel-1
    sumR <- sumR + histogramR(i)
    sumG <- sumG + histogramG(i)
    sumB <- sumB + histogramB(i)
    kumulatifHistogramR(i) <- sumR
    kumulatifHistogramG(i) <- sumG
    kumulatifHistogramB(i) <- sumB
endfor
#pragma omp parallel
for i=0 to N-1
    for j=0 to M-1
        kR <- inputR(i,j)
        kG <- inputG(i,j)
        kB <- inputB(i,j)
        outputR(i,j)<-
(GrayLevel/Area)*kumulatifHistogramR(kR)
        outputG(i,j)<-

```

```

(GrayLevel/Area) * kumulatifHistogramG(kR)
    outputB(i,j) <-
(GrayLevel/Area) * kumulatifHistogramB(kR)
    endfor
endfor

```

Penelitian tersebut menuliskan program dalam dua versi, yaitu serial dan paralel. Berikut ini hasil perbandingan yang diperoleh.

Sampel	Serial (Detik)			Parallel (detik)		
	LPF	HE	LPFHE	LPF	HE	LPFHE
Gambar 1	0.237	0.062	0.279	0.188	0.029	0.207
Gambar 2	0.568	0.218	0.692	0.556	0.087	0.826
Gambar 3	0.090	0.031	0.095	0.057	0.011	0.067

Gambar Waktu pengolahan yang diperlukan

Sampel	Speed Up		
	LPF	HE	LPFHE
Gambar 1	1.3 x	2.1 x	1.3 x
Gambar 2	1.0 x	2.5 x	0.8 x
Gambar 3	1.6 x	2.8 x	1.4 x
Rata-rata	1.3 x	2.5 x	1.2 x

Gambar Perhitungan SpeedUp

Berdasarkan tabel hasil di atas, Bastian menyimpulkan bahwa teknik pemrograman paralel menggunakan *platform* OpenMP yang diterapkan mampu mempercepat proses pengolahan citra digital dengan rata-rata sebesar 1.8x lebih cepat sehingga proses pengolahan citra digital yang dilakukan lebih menghemat waktu

3. Implementasi Algoritma Paralel pada GPU Computing

Dalam sebuah penelitian yang melakukan pengolahan data dalam jumlah yang banyak, umumnya dibutuhkan waktu yang cukup lama untuk melakukan suatu proses komputasi agar menghasilkan output yang diinginkan. Penggunaan teknologi pemrosesan paralel menjadi salah satu alternatif untuk menyelesaikan suatu tugas yang membutuhkan komputasi besar. Graphical Processing Unit (GPU) memiliki kemampuan komputasi paralel karena strukturnya yang paralel dengan banyak core yang saling bekerja sama untuk memproses piksel grafis komputer dalam volume yang besar (Sabiq, 2022).

Contoh Implementasi atau peranan algoritma paralel pada GPU computing diantaranya, yaitu:

- a. Sabiq, 2020 melakukan penelitian dengan judul sistem komputasi paralel menggunakan GPU berbasis NVIDIA CUDA. penelitian ini melakukan pengembangan sebuah sistem komputer yang digunakan untuk melakukan komputasi secara paralel dengan menggunakan GPU yang bisa diakses secara remote agar dapat dimanfaatkan dalam kebutuhan pemrosesan paralel di lingkungan Fakultas Teknologi Universitas YARSI.

Pada penelitian ini, sistem komputasi yang dibangun, digunakan GPU berbasis NVIDIA dengan chipset GeForce RTX 2070 yang memiliki 2304 Cores. CPU yang digunakan adalah intel i5-9400F 2.9GHz 6 core dengan memori DDR4 8GB. Dari hasil pengujian didapatkan GPU GeForce RTX 2070 memiliki kecepatan transfer 3 GB/s. Penelitian ini melakukan pengujian pertama dengan menguji *compiler* dan menjalankan program berbasis CUDA. Program yang diuji adalah program bandwith test dan perkalian matriks yang ada pada sampel CUDA sehingga proses kompilasi berbasis CUDA menghasilkan file *executable* bandwithTest dan hasil kompilasi bandwithTest menunjukkan bahwa program telah berhasil dijalankan menggunakan perangkat GPU GeForce RTX 2070 dengan kecepatan transfer 3 GB/s.

Proses kompilasi dan hasil dari komputasi perkalian matriks menggunakan GPU menunjukkan bahwa performa yang didapatkan dari GPU untuk melakukan komputasi pada perkalian matriks yang berukuran 320x320 yang dikalikan dengan matriks berukuran 640 x 320 adalah 816.22 GFlop/s, dan waktu yang digunakan untuk melakukan komputasi adalah 0.161 msec.

Untuk mendapatkan nilai performa dari sistem yang dibangun, penelitian ini melakukan pengujian perkalian matriks dengan ukuran 8x8 dikalikan dengan matriks berukuran 8x8 hingga perkalian matrik berukuran 4096x4096 dikali 4096x4096. Pengujian dilakukan dengan menjalankan program yang menghitung perkalian matriks yang dikomputasikan menggunakan CPU dan juga GPU untuk mendapatkan running time atau waktu yang dibutuhkan untuk melakukan perhitungan perkalian matriks pada setiap ukuran.

Hasil running time yang didapat dari komputasi menggunakan CPU dan GPU pada setiap ukuran matriks dibandingkan untuk mendapatkan nilai peningkatan kecepatan yang didapatkan dari komputasi yang menggunakan CPU ke komputasi menggunakan GPU. Dari hasil pengujian dengan perkalian matriks, pada matriks berukuran 52x52 dan lebih kecil, komputasi menggunakan CPU lebih cepat dibandingkan dengan GPU. Sedangkan pada matriks berukuran 64x64 ke atas, kecepatan komputasi GPU yang didapat lebih cepat dibandingkan CPU.

Pada perkalian matriks berukuran 4096x4096, kecepatan GPU 4312,75 kali lebih cepat dibandingkan dengan CPU. Penelitian ini juga menyarankan bahwa sistem komputasi paralel berbasis NVIDIA CUDA dapat digunakan untuk melakukan komputasi dengan data-data yang besar dan banyak, agar dapat mempersingkat waktu eksekusi program dan untuk komputasi yang menggunakan data berukuran kecil dan sedikit, tidak disarankan untuk menggunakan sistem komputer berbasis NVIDIA CUDA, karena waktu yang dibutuhkan untuk menjalankan program dapat lebih lama dibandingkan saat dieksekusi pada komputer pada umumnya yang berbasis CPU.

- b. Abd. Masjid Hamsi (2009) dalam skripsinya melakukan penelitian dengan judul komputasi paralel berbasis GPU untuk fluida tak-mampat di dalam kotak dua dimensi. Pada penelitian ini menggunakan algoritma fluida stabil, dimana algoritma fluida stabil merupakan metode yang banyak digunakan untuk simulasi fluida di dalam GPU. Metode fluida stabil hanya merupakan metode pendekatan untuk menyelesaikan persamaan Navier-Stokes. Untuk keperluan dalam bidang teknik dan sains, sehingga penelitian ini mempertimbangkan penggunaan metode fluida stabil.

Kelebihan metode fluida stabil yaitu lebih mudah diimplementasikan dalam pemrograman GPU dan lebih stabil. Kesetabilan metode fluida stabil memungkinkan pemilihan satuan langkah waktu yang lebih besar dari pada metode konvensional (metode beda hingga atau metode elemen hingga) sehingga proses simulasi bisa berjalan lebih cepat. Berdasarkan skenario paralel yang digunakan yaitu satu thread memproses satu data, jumlah maksimum data yang dapat diproses adalah jumlah maksimum dimensi grid dalam arah x dikali dengan dimensi blok arah x dikali jumlah maksimum dimensi grid arah y dikali dengan dimensi blok arah y. Pada penelitian ini menggunakan dimensi 16 x 16.

Untuk mengetahui kemampuan komputasi paralel pada GPU maka diperlukan pengujian kemampuan komputasi GPU. Iterasi Jacobi digunakan untuk menguji kemampuan komputasi paralel pada GPU. Pemilihan iterasi Jacobi sebagai pengujinya dikarenakan iterasi Jacobi merupakan proses yang paling banyak memakan waktu. Semakin besar dimensi grid, waktu proses komputasi iterasi Jacobi di CPU meningkat drastis. Ini dapat dijelaskan dengan menghitung jumlah looping / perulangan pada proses iterasi Jacobi. Misalkan, pada dimensi grid 128x128 dengan iterasi 100 maka jumlah perulangannya adalah : $128 \times 128 \times 100 = 1.638.400$ Sedangkan waktu komputasi iterasi Jacobi pada GPU relatif tidak berubah terhadap kenaikan besar dimensi grid. Hal ini menunjukkan kemampuan GPU dalam meningkatkan kecepatan komputasi. Kenaikan dimensi grid justru akan meningkatkan performa komputasi pada GPU.

Penelitian ini juga menunjukkan perbandingan waktu komputasi iterasi Jacobi antara CPU dan GPU. Perbandingan kecepatan komputasi antara CPU dan GPU meningkat seiring dengan kenaikan dimensi grid. Pada dimensi grid 16x16 dan 32x32 kecepatan komputasi pada CPU lebih tinggi dari pada GPU, karena pada dimensi grid 16x16 jumlah blok sama dengan satu sehingga tidak semua SM GPU bekerja. Sedangkan pada dimensi grid 32x32 jumlah blok adalah 4 yang sama dengan jumlah SM. Ini menunjukkan bahwa optimasi akan dicapai jika jumlah blok lebih besar dari jumlah SM GPU sehingga semua SM dijaga tetap dalam keadaan bekerja. Sebagai catatan, program iterasi Jacobi ini hanya menggunakan register dan memori global GPU untuk menyimpan data komputasi. Simulasi pada penelitian ini menunjukkan proses

simulasi. Dimensi kotak adalah 1x1 satuan. Fluidanya adalah sejenis oli dengan viskositas mekanik sama dengan 0.000194 satuan.

Proses simulasi dimulai dengan memberikan medan kecepatan mula-mula sama dengan nol. Gangguan diberikan dengan menggerakkan mouse komputer di permukaan fluida. Keadaan mula-mula fluida menunjukkan hubungan antara iterasi Jacobi dengan kecepatan simulasi (dihitung dalam frames per second/FPS). Salah satu masalah dalam animasi aliran fluida adalah jumlah frame yang dibutuhkan untuk mendapatkan pandangan yang jelas terhadap pola aliran fluida dengan FPS yang dibutuhkan untuk mendapatkan pola aliran fluida yang jelas minimal 25 FPS dan nilai FPS diatas 25 didapatkan pada saat iterasi sama dengan 5 dan akurasi yang baik diperoleh jika jumlah iterasi antara 40 sampai 80. Iterasi dibawah 20 tidak direkomendasikan karena nilai error-nya cukup besar. Sedangkan iterasi antara 30 sampai 80 nilai FPS yang diperoleh cukup kecil sehingga simulasi tidak berjalan mulus. Pilihan yang terbaik adalah menggunakan iterasi 20 atau 25. Dari percobaan simulasi, nilai iterasi iterasi 20 atau 25 sudah cukup memadai walaupun FPS-nya dibawah 25, hal ini mengakibatkan gerakan fluida tidak begitu mulus.

Dari percobaan simulasi ditemukan bahwa kecepatan fluida pada saat simulasi mula-mula tidak benar-benar sama dengan nol (walaupun kecepatan fluida sudah di-setting sama dengan nol), terutama pada daerah batas sebelah bawah. Hal ini disebabkan masih adanya bug pada program komputer yang berasal dari manajemen memori GPU yang masih kurang baik, mengingat manajemen memori merupakan aspek yang sangat penting pada pemrograman GPU. Bug ini juga berasal dari skenario paralel yang digunakan yaitu satu thread memproses satu data. Hal ini mengakibatkan kecepatan komputasi tidak terlalu optimal. Hal-hal yang bisa dilakukan untuk meningkatkan performa komputasi paralel pada GPU yaitu :

1. Coalesced memory access yaitu metode untuk meningkatkan bandwidth akses memori global.
2. Menggunakan shared memory sebagai tempat menyimpan data sementara.
3. Jumlah blok thread harus lebih besar atau sama dengan 43 jumlah multi prosessor sehingga tidak ada multiprosesor dalam keadaan idle

- (menganggur). Adanya multiprosesor dalam keadaan idle bisa menurunkan performa komputasi secara drastis.
4. Satu thread memproses lebih dari satu data sehingga jumlah data yang diproses tidak bergantung pada maksimum dimensi grid CUDA.

Penutup

Kesimpulan:

1. *Python dan Matlab*

- a. Contoh aplikasi pemrograman paralel dengan Python adalah masalah perhitungan kata dan simulasi *N-body*. Performa pada tiap kasus bergantung pada *node* maupun algoritma yang digunakan.
- b. Contoh aplikasi pemrograman paralel dengan Matlab adalah DIC. Kecepatan algoritma bergantung pada *pixel* gambar maupun kapasitas komputer yang digunakan.

2. OpenMP

OpenMP adalah *Application Program Interface (API)* untuk menulis aplikasi *multithreaded* (MT), berupa satu set direktif *compiler* dan *library* untuk pemrograman aplikasi paralel yang menyederhanakan penulisan program pada C++, C, Fortran. OpenMp lebih efisien dan memungkinkan kode paralel tingkat rendah, namun OpenMP menyembunyikan detail tingkat rendah dan memungkinkan programmer untuk mendeskripsikan kode paralel dengan konstruksi tingkat tinggi sesederhana mungkin. OpenMP memiliki arahan yang memungkinkan programmer untuk menentukan: daerah paralel, apakah variabel di bagian paralel bersifat pribadi atau bersama, bagaimana jika utas disinkronkan, cara memparalelkan loop, dan bagaimana pekerjaan dibagi antara utas (penjadwalan). Salah satu contoh aplikasi pemrograman paralel OpenMP dengan C adalah pada pengolahan citra.

3. GPU dengan CUDA

- a. Telah dibangun sistem komputasi paralel berbasis GPU dengan GPU NVDIA menggunakan GForce RTX2070. Dari hasil pengujian, sistem ini dapat digunakan secara remote melalui LAN untuk menjalankan komputasi paralel menggunakan GPU. Komputasi data menggunakan GPU membutuhkan waktu komputasi yang lebih singkat dibandingkan dengan CPU untuk data yang berukuran besar.

Sedangkan pada data yang berukuran kecil, komputasi menggunakan CPU membutuhkan waktu yang lebih singkat dibandingkan menggunakan GPU.

- b. Penggunaan iterasi Jacobi dalam menyelesaikan persamaan linear yang dihasilkan dari diskritisasi persamaan difusi dan persamaan Poisson sudah cukup memadai dalam menghasilkan simulasi yang realistik.
- c. Optimasi eksekusi di dalam GPU akan tercapai jika jumlah blok sama atau lebih besar dari jumlah multiprosesor GPU.
- d. Kecepatan komputasi iterasi Jacobi pada GPU lebih cepat dari CPU, bergantung pada besar dimensi grid.

Daftar Pustaka

- Abd Majid Hamsi. (2009). *Komputasi paralel berbasis gpu untuk fluida tak-mampat di dalam kotak dua dimensi*. Sarjana thesis, Universitas Brawijaya.
- Bastian, A, Zaliluddin, D, & Maroghi, M.S.A. 2022. IMPLEMENTASI PEMROGRAMAN PARALEL MENGGUNAKAN PLATFORM OPENMP PADA CITRA DIGITAL DENGAN METODE LOW-PASS FILTER DAN HISTOGRAM EQUALIZATION. *Infotech Journal*, 8(1): 28-33. diakses dari <https://doi.org/10.31949/infotech.v8i1.1878> tanggal 1 Desember 2022.
- Mohr, dkk. 2001. Design and Prototype of a Performance Tool Interface for OpenMP. *Journal of Supercomputing*. diakses dari https://www.researchgate.net/publication/2380669_Design_and_Prototype_of_a_Performance_Tool_Interface_for_OpenMP tanggal 11 Oktober 2022
- Sabiq, A., Yugaswara, H., & Wicaksono, H. (2021). SISTEM KOMPUTASI PARALEL MENGGUNAKAN GPU BERBASIS NVIDIA CUDA. *Orbith: Majalah Ilmiah Pengembangan Rekayasa dan Sosial*, 17(2), 158-164.
- Tim Matson. _____. *OpenMP in a nutshell*. diakses dari <https://tildesites.bowdoin.edu/~ltoma/teaching/cs3225-GIS/fall17/Lectures/openmp.html> tanggal 1 Desember 2022
- Lisandro D. Dalcin; Rodrigo R. Paz; Pablo A. Kler; Alejandro Cosimo (2011). *Parallel distributed computing using Python*. , 34(9)
- Lescisin, Michael; Mahmoud, Qusay H. (2017). [IEEE 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE) - Windsor, ON, Canada (2017.4.30-2017.5.3)] 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE) - DCM: A Python-based middleware for parallel processing applications on small scale devices. , (), 1–5.
- Jiang, Z.; Gao, Y.; Liu, J. A lid approach for predicting wave induced motions of trimaran in regular waves. Brodogradnja 2019, 70, 171–185
- Link video dan ppt:
https://univindonesia-my.sharepoint.com/:w/g/personal/elizabeth_lilies_office_ui_ac_id/ETmWOzYRQhBF13tRmVV0iBoBbV-DM3jQv5RzitgJqTEgPA?e=IwWz8t

HIGH PERFORMANCE COMPUTING

Elizabeth Lilies Megawati

ABSTRAK

Kebutuhan pada proses komputasi yang besar seperti pengolahan big data, prediksi cuaca, visualisasi 3D, dan lain-lain akan berat jika dikerjakan dengan sistem *single computer*. HPC merupakan solusi untuk mengerjakan komputasi yang besar.

HPC adalah teknologi yang menggabungkan beberapa komputer untuk bekerja sama menyelesaikan suatu pekerjaan. Terdapat tiga node pada sistem HPC, yaitu node jaringan, node komputasi, dan node penyimpanan. Empat manfaat HPC, yaitu kinerja yang membuat pekerjaan lebih efisien, menawarkan ketersediaan yang lebih tinggi dan lebih banyak waktu aktif, lebih hemat biaya, dan dapat menyelesaikan dalam sepersekian waktu yang dibutuhkan. Beberapa penggunaan HPC yaitu pada simulasi cuaca dan pencegahan bencana, kendaraan otonom, energi, virtual dan augmented reality, eksplorasi luar angkasa, fisika kuantum, dan lainnya.

Kata kunci: Komputasi Paralel, *Central Processing Unit*, *Graphics Processing Unit*, *High Performance Computing*

A. PENDAHULUAN

Metode komputasi sekuensial melibatkan operasi perangkat lunak pada satu komputer (dengan satu unit pemrosesan pusat). CPU menggunakan serangkaian instruksi untuk memecahkan masalah secara sekuensial, namun dalam satu waktu hanya satu instruksi yang dapat dieksekusi. Komputasi paralel merupakan evolusi dari metode eksekusi berurutan ini, dengan tujuan mensimulasikan keadaan alam, di mana peristiwa terjadi secara bersamaan dan dapat diproses secara paralel pada waktu yang sama.

Pada prinsipnya, komputasi paralel dapat mengurangi waktu komputasi dan mempercepat pemrosesan instruksi yang kompleks. Suatu penelitian yang dilakukan Rahmatullah berjudul *Analisis Perbandingan Performa Pemrograman Sekuensial dan Paralel dengan Skema Uji Matrix, Filter dan Quick Sort* menunjukkan bahwa performa

komputasi paralel yang telah diuji menghasilkan nilai yang lebih baik dibandingkan proses sekuenzial dengan rata-rata nilai sebesar 3.6 [Januarianto, 2012]. Proses berkisar 4x

Penelitian oleh Januarianto berjudul *Analisis Perbandingan Komputasi Sekuensial dan Komputasi Paralel GPU Memanfaatkan Teknologi NVIDIA CUDA pada Aplikasi Kompresi Citra Menggunakan Algoritma DCT 8x8* menunjukkan bahwa proses kompresi citra pada komputasi paralel lebih cepat daripada proses komputasi citra pada komputasi sekuenzial dengan rata-rata nilai sebesar 3.6 [Januarianto, 2012].

Komputasi paralel merupakan komputasi dengan metode menggunakan beberapa sumber daya komputasi pada saat yang sama (banyak yang instruksi dieksekusi secara bersamaan) untuk memecahkan masalah komputasi [Gigabyte.com]. Konsep dasarnya adalah membagi instruksi-instruksi menjadi bagian terpisah, dan kemudian menyelesaikan masalah dengan menggunakan beberapa metode eksekusi. Komputasi paralel sudah ada sejak akhir 1950 dan awal 1960. Komputasi paralel terbagi menjadi komputasi paralel *shared-memory*, komputasi paralel *distributed-memory*, dan komputasi paralel *hybrid-memory*.

Kemajuan teknologi menyebabkan munculnya kartu grafis distrik atau khusus, yaitu GPU. GPU (unit pemrosesan grafis) dan CPU (unit pemrosesan pusat) sama-sama prosesor, namun perbedaannya sendiri adalah GPU menggunakan struktur paralel yang lebih sederhana dan lebih khusus, yang memungkinkan mereka memiliki jumlah inti yang lebih banyak daripada CPU yang lebih kompleks. (dalam Intel) CPU bersifat umum dan dapat menangani hampir semua tugas komputasi, namun GPU sangat baik dalam melakukan beberapa tugas tertentu dengan sangat cepat seperti menghasilkan grafik komputer pada layar. Penelitian yang dilakukan Farisi dkk (2017) menunjukkan hasil bahwa perangkat lunak dengan menggunakan pemrosesan paralel GPU dan CPU memiliki kecepatan hampir dua kali lipat dibandingkan dengan perangkat lunak yang hanya menggunakan pemrosesan komputasi CPU saja. Penelitian oleh Kurniawan dkk (2015) juga menunjukkan Teknologi CUDA untuk komputasi paralel menggunakan GPU lebih unggul dibandingkan komputasi sekuenzial menggunakan CPU.

Kartu grafis mendukung GPU dengan memori khusus, *heatsink*, dan komponen lainnya, itulah sebabnya banyak jenis GPU ini umumnya mengungguli GPU terintegrasi yang

terpasang di motherboard atau CPU . Untuk itulah, kartu grafis menjadi andalan industri esports dan game. [[Gigabyte.com](#)]

Dalam beberapa tahun terakhir, para ilmuwan dan insinyur telah menemukan bahwa banyak tugas komputasi lainnya dapat dilakukan oleh GPU, selama data dalam bentuk grafis. GPU yang digunakan untuk sesuatu selain menghasilkan grafik kadang-kadang disebut GPGPU.

Sering berkembangnya kebutuhan manusia terkait teknologi, dibutuhkan komputer dengan komputasi yang tinggi untuk melakukan desain dan perhitungan yang kompleks, sehingga ditemukan arsitektur ***High Performance Computing (HPC)***. HPC merupakan salah satu (selain pengembangan AI) yang merupakan terobosan luar biasa yang telah dimungkinkan menggunakan server yang memanfaatkan sejumlah besar GPGPU.

B. PEMBAHASAN

Dalam makalah ini, akan dipaparkan pengenalan yang berkaitan HPC diantaranya meliputi definisi HPC, manfaat HPC, Penggunaan HPC, komponen sistem HPC, dan optimasi HPC cluster.

1. Definisi HPC

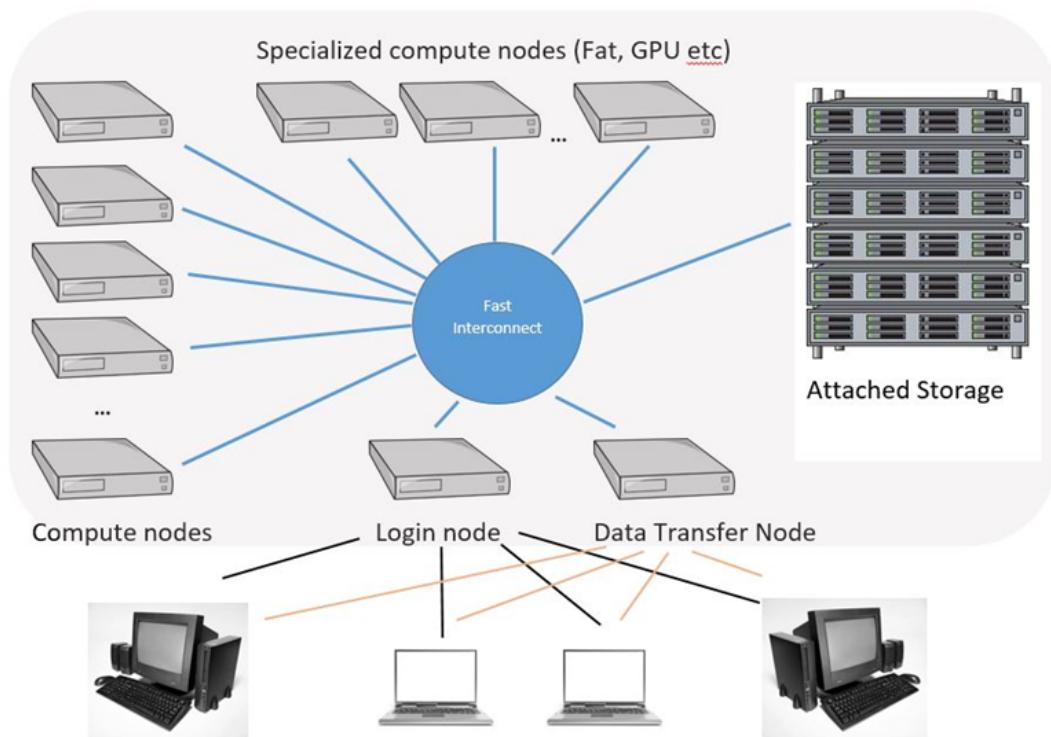
Dr. Wisnu [[dalam IPB, 2022](#)] mengungkapkan bahwa HPC adalah teknologi yang dapat memecahkan masalah kompleks yang membutuhkan komputasi yang masif dengan memanfaatkan kekuatan superkomputer. Tiga desain HPC yang biasa digunakan, yakni komputasi paralel, komputasi klaster serta komputasi grid dan terdistribusi. Pemanfaatan High Performance Computing (HPC) atau Komputasi Berperforma Tinggi menjadi salah satu terobosan saintifik untuk membantu peneliti dalam mengeksplorasi data. HPC ini merupakan kebutuhan penting pada era dimana data menjadi paradigma keempat dalam sains. Kemudahan eksplorasi dan manipulasi data kini menjadi semakin memungkinkan. [[IPB](#)]

Komputasi HPC menyelesaikan masalah yang rumit di dunia nyata dengan cara yang inovatif menggunakan kemampuan fleksibel yang dapat diskalakan dari prosesor-prosesor yang dapat diskalakan, alat perangkat lunak, penyimpanan, dan memori. [[Intel.co.id](#)]

HPC merujuk luas pada kategori komputasi canggih yang dapat menangani jumlah data yang lebih besar, melakukan serangkaian perhitungan yang lebih kompleks, dan berjalan pada kecepatan yang lebih tinggi daripada komputer pribadi rata-rata. Beberapa item HPC paling terkenal di dunia berfungsi pada skala superkomputer dari *quadrillion floating point operations* per detik (petaFLOPS), atau bahkan *quintillion floating point operations* per detik (exaFLOPS). [Gigabyte.com]

2. Komponen Sistem HPC

Pada umumnya HPC terdiri dari beberapa komputer yang bekerja secara individual. Masing-masing komputer terhubung melalui sebuah jaringan dengan satu atau beberapa komputer sebagai pengatur.



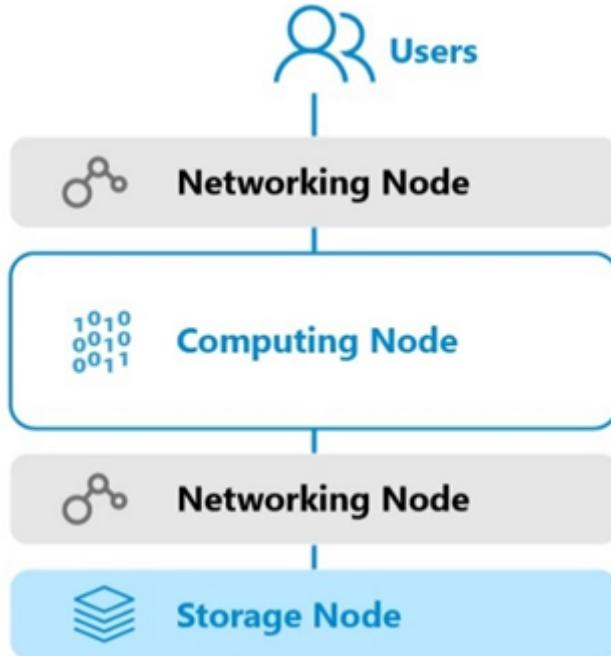
Gambar 1 Ilustrasi Sistem HPC

Sumber: [www.hpc.iastate.edu]

Masing-masing komputer disebut sebagai *node*, di dalam sebuah HPC ada kemungkinan terdiri dari node dengan spesifikasi yang beraneka ragam, yaitu: [Gigabyte.com]

- a. Node jaringan, yaitu menghubungkan atau membantu server berbicara satu sama lain di dalam sistem, dan menghubungkan sistem ke dunia luar.
- b. Node komputasi, yaitu otak dari seluruh operasi mereka melakukan perhitungan dan penyelesaian tugas.

- c. Node penyimpanan, yang menyimpan semua data baik permintaan maupun tanggapan.



Gambar 2 Node pada HPC

Sumber: [www.gigabyte.com]

Saat mengakses HPC, pengguna tidak dapat melakukan akses langsung terhadap node, pengguna hanya dapat berinteraksi terhadap konstroler HPC melalui akses SSH. Setiap node mempunyai minimal satu buah prosesor, pada tiap prosesor pada umumnya terdiri dari minimal 1 buah core. Masing-masing core mempunyai *Floating Point Unit* (FPU) yang bertanggung jawab melakukan proses komputasi data. [binus.ac.id]

Kinerja prosesor sangat penting untuk mencapai HPC. Dua CPU besar, yaitu AMD dan Intel, sedang berlomba untuk menghasilkan prosesor yang lebih banyak. Sebuah alternatif untuk arsitektur x86 adalah arsitektur ARM, yang menggunakan desain yang sangat berbeda dimana memungkinkan satu prosesor untuk menampung lebih banyak inti. Konsumsi daya prosesor ARM yang lebih rendah juga membuat manajemen termal lebih mudah. Pada akhirnya, jenis prosesor yang dipasang di sistem HPC sangat bergantung pada tugas yang dirancang untuk sistem tersebut. [Gigabyte.com]

Selain CPU, akselerator GPU (atau GPGPU) yang dipasang di node komputasi juga dapat menjadi pengubah permainan, juga cocok untuk komputasi paralel, komputasi grid, dan bentuk komputasi terdistribusi lainnya. Dengan kata lain, jika tugas yang ada dioptimalkan dengan bekerja bersama akselerator, pengguna dapat mencapai HPC tidak

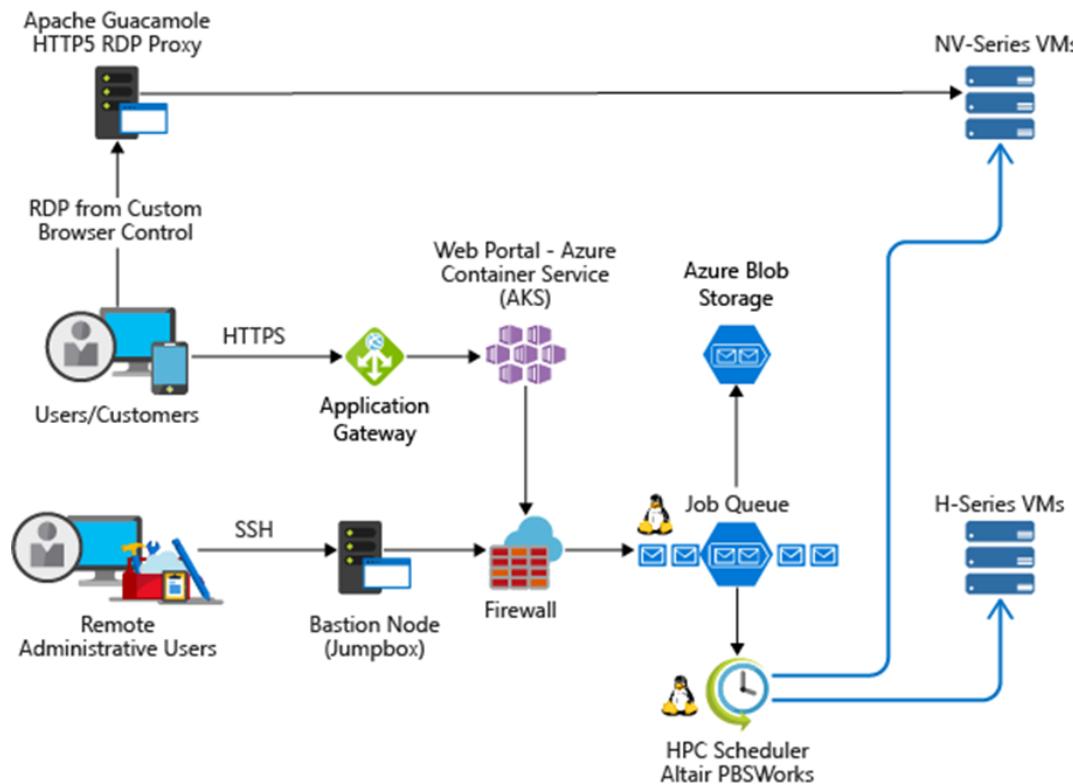
hanya dengan membeli CPU yang kuat, tetapi dengan menambahkan GPU yang tepat. [Gigabyte.com]

Seperti disebutkan, ketersediaan tinggi adalah aturan praktis dalam hal HPC tidak hanya sistem harus aktif sesering mungkin, data yang disimpan dalam sistem HPC juga harus aman dan dapat diakses. RAID (*Redundant Array of Inexpensive Disks*), metode virtualisasi yang menggabungkan beberapa disk menjadi satu unit untuk mencapai redundansi, sering digunakan pada node penyimpanan. Standar antarmuka yang dikenal sebagai NVMe (*Non-Volatile Memory Express*) mempercepat transfer data, sehingga perhitungan di node komputasi tidak terhambat oleh kecepatan transfer yang tidak memadai. Singkatnya, meskipun node penyimpanan mungkin bukan titik fokus sistem HPC, mengadopsi teknologi penyimpanan yang benar dapat sangat membantu dalam meningkatkan kinerja sistem secara keseluruhan. [Gigabyte.com]

Node jaringan terdiri dari server yang dirancang untuk meng-*host* intranet, menyediakan akses melalui VPN, atau berbagi koneksi ke internet. Hal tersebut dapat dilengkapi dengan perangkat lain, seperti saklar, *router*, *firewall*. Penerapan standar jaringan terbaru seperti Ethernet dan InfiniBand (IB), dapat memastikan bahwa data ditransfer dengan cepat dan aman tidak hanya antara sistem HPC dengan penggunanya, tetapi juga di antara berbagai node di dalam sistem HPC. [Gigabyte.com]

Selain perangkat keras, komponen perangkat lunak dari sistem HPC sama pentingnya. Perangkat lunak platform HPC, kerangka kerja yang dioptimalkan, pustaka, dan alat lainnya akan menjamin pengguna mendapatkan hasil maksimal dari sistem HPC. Kesesuaian adalah kuncinya, jika solusi perangkat keras dan perangkat lunak cocok untuk tugas si pengguna, maka pengguna akan menuai manfaat penuh dari HPC. [Gigabyte.com]

Berikut ini merupakan contoh skenario solusi HPC dibangun dengan bantuan komputer di Azure. [learn.microsoft.com]

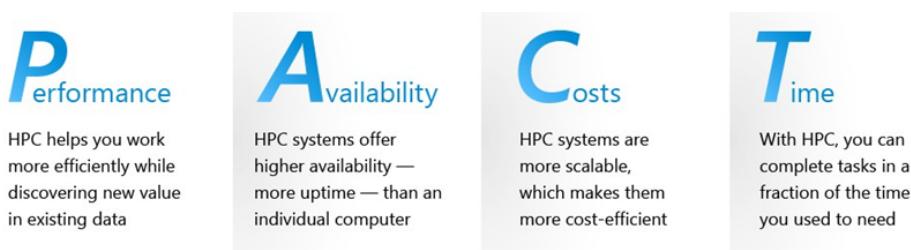


Gambar 3 Skenario solusi HPC di Azure

Sumber: learn.microsoft.com

3. Manfaat HPC

Berikut ini empat manfaat yang dapat diingat dengan singkatan praktis PACT, yaitu:
(*Gigabyte.com*)



Gambar 4 Manfaat HPC

Sumber: [gigabyte.com]

a. **Performance** (Kinerja)

Sebuah perusahaan dengan akses ke HPC akan mengungguli persaingan dilihat dari melakukan tugas yang lebih efisien dan nilai kebaruan yang dapat diperoleh dari data yang ada, contohnya adalah dalam upaya yang dilakukan untuk merancang produk baru seperti mobil atau seperangkat peraturan baru, seperti pedoman yang

membantu bandara dalam mencegah penundaan jadwal penerbangan. Sebelum menggunakan HPC, banyak pengujian fisik, belum lagi coba-coba, terlalu banyak bagian yang bergerak terlibat namun dengan menjalankan simulasi pada sistem HPC selama pengembangan, produk atau prosedur baru akan memiliki peluang yang jauh lebih baik untuk diterapkan.

Sistem HPC juga dapat menyaring data besar yang ada untuk mencari nilai yang sampai sekarang belum ditemukan. Salah satu contohnya, kecerdasan buatan (AI) digunakan dalam analisis rekam medis untuk menemukan indikator penyakit. Dalam konteks tersebut, HPC tidak hanya membantu dokter bekerja lebih efisien, tetapi juga menyaring data yang ada untuk mencari nilai baru.

b. ***Availability*** (Ketersediaan)

Ketersediaan mengacu pada konsep “ketersediaan tinggi”, yang berarti peralatan IT harus menawarkan waktu aktif sebanyak mungkin agar pengguna dapat memanfaatkan layanan sepenuhnya. Ini adalah kebutuhan dasar di era digital karena semakin banyak kehidupan kita yang berputar di sekitar penemuan teknologi.

Ketersediaan sistem HPC umumnya lebih unggul daripada satu buah komputer karena node dari sistem HPC terdiri lebih dari satu komputer atau server dan node dirancang untuk bekerja bersama dan saling mendukung.

c. ***Cost*** (biaya)

Sistem pada HPC lebih terukur karena komputasi tersebar di berbagai sumber daya. Pengguna dapat meningkatkan sumber daya (seperti CPU, GPU, memori, atau sumber daya lainnya) dan memperluas (menambahkan lebih banyak node ke dalam klaster) saat dibutuhkan, daripada membeli lebih banyak peralatan dibanding yang diperlukan di awal sebagai antisipasi dari perkembangan mendatang. Menyewa sumber daya HPC dari penyedia layanan *cloud* (CSP) dapat lebih meningkatkan skalabilitas dan menurunkan biaya.

d. ***Time*** (waktu)

Rata-rata PC tingkat konsumen berfungsi pada tingkat gigaFLOPS (satu juta FLOPS) atau teraFLOPS (satu miliar FLOPS). Tetapi seperti yang telah ditetapkan, sistem HPC diukur pada skala petaFLOPS, atau bahkan exaFLOP, yang lebih cepat. HPC mampu menyelesaikan perhitungan dalam hitungan menit atau jam, bukan hari atau bulan.

Penelitian yang dilakukan oleh Louie (2018) berjudul *Strategies of data layout and cache writing for input-output optimization in high performance scientific*

computing: Applications to the forward electrocardiographic problem menunjukkan bahwa adanya peningkatan yang signifikan dalam kinerja I/O aplikasi HPC di berbagai disiplin ilmu, mengurangi waktu eksekusi dan pasca pemrosesan, dan mengarah pada penggunaan sumber daya HPC yang lebih efisien.

Dalam JawaPos.com (2019), pihak LG mengungkapkan bahwa melalui HPC dan GPU yang didukung Microsoft Azure, teknologi LG akan secara drastis dapat mengurangi waktu belajar dan berkembang yang dibutuhkan perangkat lunak AI bagi berkendara otonom milik LG.

4. Penggunaan HPC

Setelah melihat seluruh manfaat HPC, berikut ini sekilas tentang penggunaan HPC di berbagai sektor [[Gigabyte.com](#)]

a. Simulasi Cuaca dan Pencegahan Bencana

Perubahan iklim mempengaruhi pola cuaca di seluruh dunia, seperti adanya bencana alam seperti tsunami dan gelombang badai menjadi lebih berbahaya. Ada kebutuhan mendesak untuk mempelajari cuaca ekstrem dan merekayasa kota-kota untuk menahan bencana alam yang mungkin terjadi.

Dalam situs BMKG [[Thirafi, 2021](#)], Kepala BMKG Dwikorita Karnawati mengungkapkan bahwa BMKG akan berencana mengimplementasikan HPC dengan skala lebih dari 2 PetaFlops, sehingga menjadikan sistem peringatan dini BMKG jauh lebih cepat, tepat, dan akurat dalam menganalisis berbagai kompleksitas dan ketidakpastian dalam fenomena cuaca, iklim, tektonik dan kegununganapian.

b. Kendaraan Otonom

Salah satu penemuan paling menarik di cakrawala adalah kendaraan otonom. Visi komputer dan metode pembelajaran mendalam digunakan untuk “melatih” AI mobil untuk mendeteksi, mengenali, dan bereaksi terhadap kondisi jalan-seperti halnya pengemudi manusia. Sistem HPC sangat penting untuk pelatihan algoritma *self-driving* yang efektif.

Franchi dkk melakukan penelitian berjudul *A Parallel Autonomous Vehicle Simulation Pipeline on High-performance Computing*. Penelitian tersebut menyajikan kerangka kerja simulator AV pada HPC. Simulator Av yang dimaksud adalah simulator kendaraan otonom yang terintegrasi dengan simulator lalu lintas

mikroskopis. Penelitian tersebut dapat berfungsi sebagai platform untuk upaya simulasi robotika baru, juga menjadi panduan penting bagi peneliti dan praktisi kendaraan otonom dalam mensimulasikan kendaraan otonom secara efisien dan efektif dalam arus lalu lintas jalan raya campuran. [Franchi, 2021]

c. Energi

Sektor energi, termasuk industri minyak dan gas, mulai menggunakan HPC untuk mencari lokasi ekstraksi baru. Gambar 2D dan 3D kompleks yang dikumpulkan selama survei geologis yang dapat dianalisis dengan cepat dan akurat dengan HPC untuk menentukan lokasi pengeboran yang paling sesuai, sehingga akan mengurangi biaya eksplorasi.

Xue He menggunakan HPC dalam penelitiannya yang berjudul *Simulation and verification in high-performance computing for cluster distributed doubly fed induction generators in the horizon of Ecological marxism*, dimana penelitian tersebut menggunakan mode pembangkit listrik tenaga angin terdesentralisasi berdasarkan energi bersih untuk menggantikan tenaga panas tradisional yang terpusat di Cakrawala Marxisme Ekologi sebagai tujuan mengurangi emisi karbon untuk menyelesaikan krisis ekologi global. [Xue He, 2021]

d. Virtual dan Augmented Reality

Virtual Reality (VR), *Augmented Reality* (AR), dan *Mixed Reality* (MR) telah menjadi bagian dari kehidupan kita sehari-hari. Sekarang setelah raksasa teknologi internasional telah memberikan perhatiannya pada pembuatan “Metaverse”, ada baiknya melihat bagaimana HPC dapat membantu.

Hojny dkk melakukan penelitian berjudul *Application of Virtual Reality and High Performance Computing in Designing Rotary Forming Processes*. Dalam penelitiannya, diperoleh bahwa solusi sistem VR-HPC yang dikembangkan memungkinkan proses manufaktur direkayasa dan dikendalikan secara efektif dalam kondisi industri. [Hojny, 2022]

e. Eksplorasi Luar Angkasa, Fisika Kuantum, dan Lainnya

Banyak terobosan ilmiah sangat bergantung pada HPC karena HPC merupakan salah satu cara tercanggih untuk menghitung.

Penelitian oleh Kary yang berjudul *BioinfoPortal: A scientific gateway for integrating bioinformatics applications on the Brazilian national high-performance computing network* menghasilkan bahwa aplikasi HPC menyajikan bagaimana *machine learning* diterapkan untuk mengoptimalkan fungsionalitas BioinfoPortal

berdasarkan rekomendasi model prediktif untuk alokasi sumber daya yang efisien yang diperoleh lebih dari 75% efisiensi kinerja. [Kary, 2020]

Libing Zhu melakukan penelitian berjudul *Monte Carlo performance study of virtual high performance computing cluster over cloud*. Dalam penelitian tersebut, beliau merancang dan mengembangkan sistem vHPC untuk pekerjaan dalam bidang teknik nuklir, kemudian hasilnya adalah vHPC menjadi alternatif yang baik untuk digunakan di bidang teknik nuklir karena membuat HPC menjadi fleksibel dan mudah digunakan. [Libing Zhu, 2022]

C. PENUTUP

1. Kesimpulan

- a. HPC adalah teknologi penggabungan beberapa komputer yang bekerjasama menyelesaikan masalah suatu pekerjaan yang membutuhkan performa komputasi tinggi.
- b. Terdapat tiga node pada sistem HPC, yaitu node jaringan, node komputasi, dan node penyimpanan.
- c. Empat manfaat HPC, yaitu kinerja yang membuat pekerjaan lebih efisien, menawarkan ketersediaan yang lebih tinggi dan lebih banyak waktu aktif, lebih hemat biaya, dan dapat menyelesaikan dalam sepersekian waktu yang dibutuhkan.
- d. Beberapa implementasi penggunaan HPC yaitu pada simulasi cuaca dan pencegahan bencana, kendaraan otonom, energi, virtual dan augmented reality, eksplorasi luar angkasa, fisika kuantum, dan lainnya.

2. Saran

Penelitian dengan HPC masih menghadapi berbagai tantangan yakni terkait biaya pengadaan dan pengelolaan, keamanan data, tata kelola data, transfer data, dan performa dari HPC. Teknologi ini tentu harus dimanfaatkan sebagai mungkin demi pengembangan ilmu pengetahuan alam di Indonesia. Beberapa perguruan tinggi juga memiliki infrastruktur HPC. Semoga lebih banyak lagi peneliti-peneliti di Indonesia yang membahas tentang HPC sebagai terobosan saintifik yang dapat membantu peneliti mengeksplorasi data dengan tujuan mengembangkan sumber daya alam di Indonesia.

Daftar Pustaka

Badan Riset dan Inovasi Sosial. 2022. *Tren Riset dalam Sistem Transportasi dan Pengenalan HPC* BRIN. diakses dari <https://www.brin.go.id/news/110436/trend-riset-dalam-sistem-transportasi-dan-pengenalan-hpc-brin> tanggal 25 Oktober 2022

Bandung Technologi Zone. _____. KOMPUTASI PARALEL DENGAN HPC CLUSTER. diakses dari <https://bandungtechnologyzone.co.id/index.php/layanan/parallelcomputing> diakses tanggal 24 Oktober 2022

Binus University. _____. klaster dan Komputasi Paralel. diakses dari <https://binus.ac.id/malang/2020/07/kluster-dan-komputasi-paralel/> diakses tanggal 24 Oktober 2022

Birra, Fadhil Al. 2019. Kerja Sama Bareng Microsoft, LG akan Buku Industri Kendaraan Otonom. diakses dari <https://www.jawapos.com/oto-dan-tekno/teknologi/10/01/2019/kerja-sama-bareng-microsoft-lg-akan-buka-industri-kendaraan-otonom/> tanggal 26 Oktober 2022

FMIPA IPB. 2022. *Dr Wisnu Ananta Kusuma Jelaskan Riset dan Inovasi Para Peneliti IPB University Terkait High Performance Computing dalam Bidang Ilmu Pengetahuan Alam* <http://fmipa.ipb.ac.id/dr-wisnu-ananta-kusuma-jelaskan-riset-dan-inovasi-para-peneliti-ipb-university-terkait-high-performance-computing-dalam-bidang-ilmu-pengetahuan-alam/>

Franchi, Matthew & Kahn, Rebecca & Ngo, Linh & Khan, Sakib & Chowdhury, Mashrur & Kennedy, Ken & Apon, Amy. 2021. PAVSP: A Parallel Autonomous Vehicle Simulation Pipeline on High-performance Computing.

Gigabyte. _____. GPU. diakses dari <https://www.gigabyte.com/Glossary/gpu> diakses tanggal 24 Oktober 2022

Gigabyte. _____. *Parallel Computing*.

<https://www.gigabyte.com/Glossary/parallel-computing> diakses tanggal 24 Oktober 2022

He Xue. 2021. Simulation and verification in high-performance computing for cluster distributed doubly fed induction generators in the horizon of Ecological Marxism. *2021 International Conference on Energy Engineering and Power System (EEPS2021)*, 7, 14-21.

Hojny, M., Marynowski, P., Lipski, G., Gądek, T., & Nowacki, Ł. 2022. Application of virtual reality and high performance computing in designing rotary forming processes. *Archives of Metallurgy and Materials*, 67(No 3), 1099-1105.

Januariano, Andika, & Adang S. 2012. Analisis Perbandingan Komputasi Sekuensial dan Komputasi Paralel GPU Memanfaatkan Teknologi NVIDIA CUDA pada Aplikasi Kompresi Citra Menggunakan Algoritma DCT 8x8. diakses dari https://www.academia.edu/5274592/ANALISIS_PERBANDINGAN_KOMPUTASI_SEKUENSIAL_DAN_KOMPUTASI_PARALEL_GPU_MEMANFAATKAN_TEKNOLOGI_NVIDIA_CUDA_PADA_APLIKASI_KOMPRESI_CITRA_MENGGUNAKAN_ALGORITM_A_DCT_8X8_1_Andika_Januarianto_50407094_ tanggal 28 Oktober 2022

Kary A.C.S. Ocaña, Marcelo Galheigo, Carla Osthoff, Luiz M.R. Gadelha, Fabio Porto, Antônio Tadeu A. Gomes, Daniel de Oliveira, & Ana Tereza Vasconcelos. 2020. BioinfoPortal: A scientific gateway for integrating bioinformatics applications on the Brazilian national high-performance computing network. *Future Generation Systems*, 107, 192-214.

Libing Zhu, Ze Xi, Peng Cong, Gongyi Yu, Yuan Liu, Xincheng Xiang, Wei Xu, Xiangang Wang. 2022. Monte Carlo performance study of virtual high performance computing cluster over cloud. *Radiation Medicine and Protection*, volume 3, issue 3, 108-114

Louie Cardone-Noott, Rodriguez, B., & Alfonso Bueno-Orovio. (2018). Strategies of data layout and cache writing for input-output optimization in high performance scientific computing: Applications to the forward electrocardiographic problem. *PLoS One*, 13(8).

Thirafi, Hatif. 2021. Perkuat Sistem Peringatan Dini, Super Komputer BMKG Bakal “Disuntik” Teknologi HPC terkini. diakses dari <https://www.bmkg.go.id/berita/?p=perkua> t-sistem-peringatan-dini-super-komputer-bmkg-bakal-disuntik-teknologi-hpc-terkini &lang=ID&tag=press-release tanggal 26 Oktober 2022

Rahmatullah, G.M., Andry FZ, & M. Reza H. 2021. Analisis Perbandingan Performa Pemrograman Sekuensial dan Paralel dengan Skema Uji Matrix Folter dan Quick Sort. *Teknologi Informasi Universitas Lambung Mangkurat*, 6(No 1), 19-24

Pemrograman Paralel, Taksonomi Flynn, dan Aplikasi Pemrograman Paralel

Cintya Kusuma Mahadhika

Abstract

Telah dipelajari beberapa tipe pemrosesan dalam taksonomi Flynn. Dalam makalah ini, dibahas mengenai beberapa contoh kasus dalam taksonomi Flynn. Dapat disimpulkan bahwa masing-masing pemrosesan maupun alat pemrograman memiliki algoritma maupun efisiensi masing-masing. Oleh karena itu, pembaca dapat menentukan alat pemrograman yang sesuai dengan masalah yang dimiliki. Diberikan pula contoh kasus dengan pemrograman paralel sebagai referensi.

Kata kunci: **Taksonomy Flynn, komputasi paralel, aplikasi pemrograman paralel.**

PENDAHULUAN

Seiring dengan perkembangan pemikiran manusia, teknologi kemudian berpaling dari perangkat analog menjadi perangkat digital yang multifungsi dan memiliki kehandalan yang tinggi. Pada dunia sains, komputasi memungkinkan penyelesaian problem yang tidak bisa diselesaikan melalui eksperimen tradisional maupun teoritis. Seiring dengan meningkatnya kebutuhan komputasi yang cepat dan naiknya harga peralatan komputer maka pengembangan aplikasi berbasis paralel termasuk komputasi paralel yang digunakan untuk menyelesaikan permasalahan waktu komputasi pada model komputasi standalone semakin banyak dilakukan. Pemanfaatan resource-resource komputasi yang tersedia secara paralel akan mempercepat eksekusi program. Program yang dieksekusi pada komputer master akan dibagi kepada node-node komputer untuk dimanfaatkan resourcenya. Komputasi ini sangat berguna untuk algoritma-algoritma yang memiliki tingkat pertumbuhan fungsi yang kuadratik dan eksponensial. Menggunakan konsep komputasi paralel, biaya investasi untuk komputer dapat ditekan dan konsumsi energi listrik, biaya instalasi serta pemeliharaan juga menjadi lebih rendah. Komputasi pararel fleksibel terhadap perubahan teknologi komputer yang sangat cepat. Alternatif populer saat ini adalah computer clustering (kelompok komputer) atau parallel computer (komputer paralel). Sistem ini merupakan penggabungan beberapa PC (disebut node) menjadi seolah-olah satu komputer dengan kemampuan yang lebih besar (Annas dkk, 2010).

Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer independen. Hal ini umumnya diperlukan saat

kapasitas komputasi yang diperlukan sangat besar, baik karena pengolahan data maupun karena tuntutan proses komputasi yang banyak. Pada umumnya kasus ini ditemui yakni komputasi numerik yaitu menyelesaikan persamaan matematis di bidang fisika, kimia, teknik dan bidang lainnya. Dalam melakukan aneka jenis komputasi diperlukan infrastruktur mesin paralel yang terdiri dari komputer dan jaringan (perangkat lunak pendukung yang disebut middleware yang berperan dalam mengatur distribusi pekerjaan antar node dalam mesin paralel. Teknik pemrograman komputasi paralel adalah teknik eksekusi perintah atau operasi secara simultan baik dalam komputer satu prosesor ataupun dengan banyak prosesor (Annas dkk, 2010).

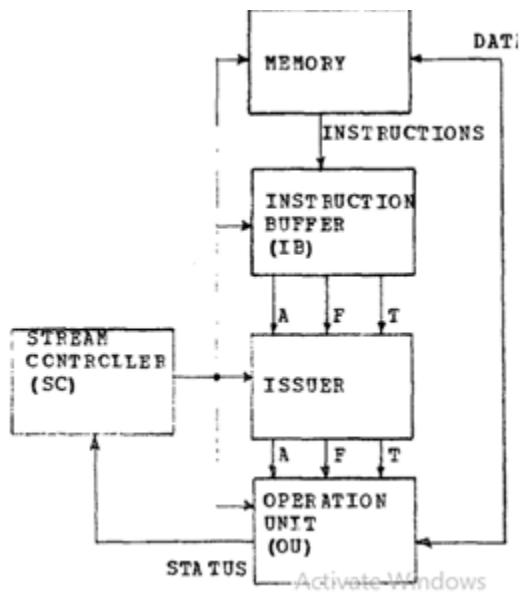
Dalam arsitektur komputer, taksonomi Flynn adalah klasifikasi yang membagi klasifikasi komputer dalam empat arsitektur, yaitu SISD (single instruction, single data), SIMD (single instruction, multiple data), MISD (multiple instruction, single data), dan MIMD (multiple instruction, multiple data) (Flynn, 1966). Masing-masing arsitektur memiliki ciri algoritma dan efisiensinya sendiri. Seperti halnya arsitektur komputer, pemrograman paralel juga memiliki tipe dan karakteristiknya sendiri. Dalam makalah ini, akan dibahas lebih lanjut mengenai studi kasus dalam taksonomi Flynn dan pemrograman paralel.

PEMBAHASAN

A. Taksonomi Flynn

1. SISD (Single Instruction, Single Data)

Rosocha dan Lee (1979) melakukan penelitian bejulul “*Performance Enhancement of SISD Processor*. Koordinasi otomatis eksekusi instruksi prosesor SISD diperiksa dalam konteks meminimalkan efek eksekusi cabang. Area, prefetch instruksi, resolusi cabang, dan organisasi penerbit diperiksa untuk kemungkinan perbaikan. Dorongan dari teknik ini adalah untuk memperluas pemrosesan bersyarat dan menggunakan pandangan ke depan untuk mendeteksi cabang. Buffer instruksi terstruktur, konvergen, digunakan untuk menyangga satu atau lebih level instruksi yang dibuat secara kondisional.



Gambar 1 kecepatan tinggi Processor SISD

```

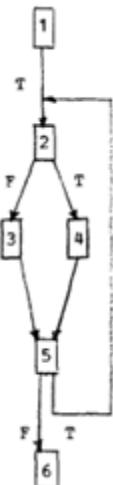
Arithmetic/Logic
<opcode><dest. reg.>;<source1><source2>
(dest. reg = source1 opcode source2)

Memory Reference
<opcode><Data reg.><address reg>
(transmit data between data register and
memory location addressed by address
register.)

Branch
<opcode><branch address>
(Evaluate the predicate specified by
the opcode. If true, activate the
block at location <branch address>;
otherwise activate the immediate
successor block.)

```

Gambar 2 Sintaks instruksi

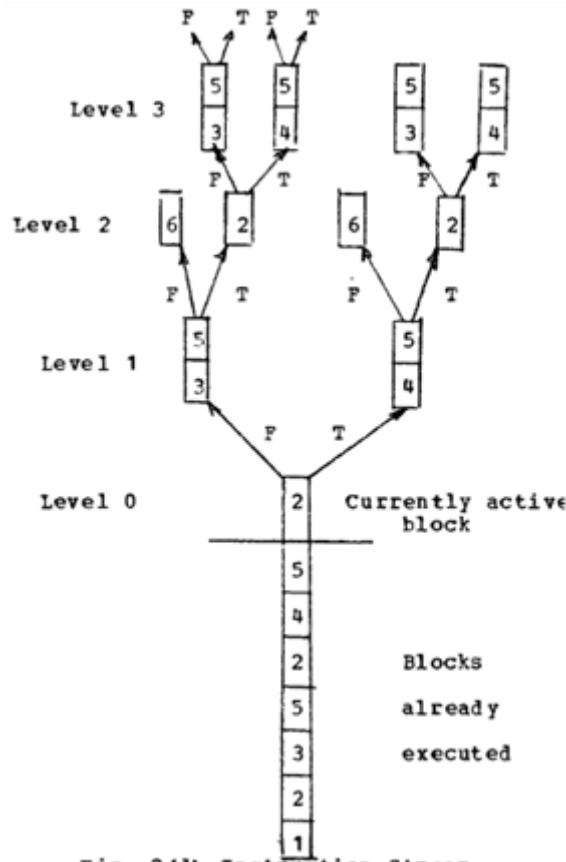


Gambar 3 struktur program statis

```
/*      BLOCK 1          */
/* Load the necessary address      */
/* registers, initialize loop      */
/* counter, index, and accumulator */
/* registers                      */
/*
1.   LD  ARB; B
2.   LD  ARC; C
3.   LD  ARD; D
4.   'etc.,'

5.   LD  RN; N
6.   LD  RI; #0
7.   LD  PSUM; #0
/*
8.   BLOCK 2          */
9.   LD  BX; INC BX
10.  Fetch B(I)        */
11.  ADD BX;ARB,PI
12.  LD  PI; PX
13.  Fetch C(I)        */
14.  ADD CY;ARC,PI
15.  LD  P2;PY
16.  Generate A        */
17.  MUL P3; PI,P2
18.  Branch if A > or = 0    */
19.  RGE POS           */
20.  R  SUM             */
/*
21.  BLOCK 3          */
22.  Fetch E(I)        */
23.  ADD BX;ARE,PI
24.  LD  PI;PX
25.  Generate D        */
26.  MUL B3;P3,PI
27.  R  SUM             */
/*
28.  BLOCK 4          */
29.  Fetch F(I)        */
30.  POS: ADD BY;ARP,RI
31.  LD  P2;PY
32.  MUL B3;P3,PI
33.  RNE LOOP          */
/*
34.  BLOCK 5          */
35.  Fetch G(I)        */
36.  SUM: ADD PX;APG,PI
37.  LD  R2;RX
38.  Generate new G(I)   */
39.  ADD B3;RN,P3
40.  ADD R3;P2,P3
41.  ST  B3; RX
42.  ADD PSUM; PSUM,P3
43.  Determine if looping completed */
44.  DEC RN             */
45.  BNE LOOP          */
/*
46.  BLOCK 6          */
47.  Store SUM          */
48.  ST  PSUM; ABS
```

Gambar 4 kode mesin dari program contoh



Gambar 5 aliran instruksi

Konvergen meningkatkan instruksi penyediaan respon dengan memberikan yang tingkat yang lebih tinggi pada instruksi prefetch. Analisis menunjukkan bahwa terdapat manfaat mempertahankan lebih dari 1 tingkat prefetch. Selain itu, konvergen dapat menyediakan aliran instruksi bersyarat untuk organisasi yang ingin mengekstrak konkurensi instruksi tambahan secara dinamis yang dihambat oleh batas blok dinamis. Skema pengkondisian akan memungkinkan tumpang tindih resolusi cabang dengan eksekusi instruksi. Meskipun mobilitas CCI yang diharapkan kurang dari tiga instruksi, pengurangan waktu aktivasi akan dihemat pada sebagian kecil dari semua eksekusi cabang. Penghematan tambahan dimungkinkan dengan mengintegrasikan evaluasi predikat dalam unit fungsi. *Issuer organization* adalah yang mempromosikan tingkat penerbitan dan eksekusi yang tinggi. Keuntungan ini diperoleh dengan mendistribusikan fungsi masalah ke fase eksekusi. Pemrosesan bersyarat dimasukkan dengan memperluas algoritma Tomasulo. Yang jelas, area penting yang mempengaruhi kinerja prosesor SISD telah diabaikan. Ini termasuk akses operan, organisasi i/o, dan strategi reorganisasi

program. Namun, manfaat dari strategi organisasi yang disajikan di sini, respons yang cepat terhadap eksekusi cabang, tampaknya berguna secara umum.

2. SIMD (Single Instruction, Multiple Data)

SIMD menggambarkan komputer dengan beberapa elemen pemrosesan yang melakukan operasi yang sama pada beberapa titik data secara bersamaan (Stokes, 2000). Arsitektur SIMD mengeksplorasi paralelisme data dengan mengeluarkan instruksi yang sama ke semua ALU (Arithmetic Logical Units) di setiap siklus. Berikut ini akan diberikan contoh algoritma dalam SIMD seperti yang dicontohkan oleh Schmidt dkk (2018), yaitu algoritma perkalian matriks $A \times B = C$ (A matriks berukuran $m \times l$, B matriks berukuran $l \times n$, C matriks berukuran $m \times n$) menggunakan program C++ sebagai berikut.

```
1 #include <cstdint>      // uint32_t
2 #include <iostream>      // std::cout
3 #include <immintrin.h> // AVX intrinsics
```

```
void plain_tmm(float * A,
               float * B,
               float * C,
               uint64_t M,
               uint64_t L,
               uint64_t N) {

    for (uint64_t i = 0; i < M; i++) {
        for (uint64_t j = 0; j < N; j++) {
            float accum = float(0);
            for (uint64_t k = 0; k < L; k++)
                accum += A[i*L+k]*B[j*L+k];
            C[i*N+j] = accum;
        }
    }
}

void avx2_tmm(float * A,
               float * B,
               float * C,
               uint64_t M,
               uint64_t L,
               uint64_t N) {

    for (uint64_t i = 0; i < M; i++) {
        for (uint64_t j = 0; j < N; j++) {

            __m256 X = _mm256_setzero_ps();
            for (uint64_t k = 0; k < L; k += 8) {
                const __m256 AV = _mm256_load_ps(A+i*L+k);
                const __m256 BV = _mm256_load_ps(B+j*L+k);
                X = _mm256_fmadd_ps(AV,BV,X);
            }

            C[i*N+j] = hsum_avx(X);
        }
    }
}
```

Apabila kode tersebut dieksekusi dalam Intel i7-6800K CPU dengan dimensi matriks

$m = 1024, l = 2048, n = 4096$, maka akan diperoleh

```
# elapsed time (avx2_tmm): 2.133s.
```

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

MISD adalah salah satu arsitektur dalam taksonomi Flynn di mana banyak unit fungsional melakukan operasi yang berbeda pada data yang sama (Flynn, 1966). MISD sangat jarang ditemukan dalam contoh nyata. Dalam beberapa literatur, disebutkan bahwa MISD digunakan dalam *systolic array*, yaitu proses di mana data input paralel mengalir melalui jaringan node prosesor terprogram, menyerupai otak manusia yang menggabungkan, memproses, menggabungkan atau mengurutkan data input menjadi hasil turunan (Flynn & Rudd, 1996). Namun, beberapa literatur juga menyebutkan bahwa *systolic array* tidak memenuhi klasifikasi MISD sebab input *systolic array* biasanya berupa vektor nilai independen serta nilai input ini digabungkan dan digabungkan ke dalam hasil dan tidak mempertahankan independensinya.

Kemudian, beberapa node tidak beroperasi pada data yang sama, yang membuat *systolic array* tidak memenuhi syarat untuk diklasifikasikan sebagai MISD. Alasan lain mengapa array sistolik tidak memenuhi syarat sebagai MISD adalah sama dengan alasan yang mendiskualifikasinya dari kategori SISD: data input biasanya berupa vektor, bukan nilai data tunggal, meskipun orang dapat berargumen bahwa vektor input apa pun yang diberikan adalah kumpulan data tunggal. (Quinn, 2004)

4. MIMD (*Multiple Instruction, Multiple Data*)

MIMD adalah suatu arsitektur komputer dalam komputasi Flynn di mana mesin memiliki sejumlah prosesor yang berfungsi secara asinkron dan independen. Setiap

saat, prosesor yang berbeda dapat mengeksekusi instruksi yang berbeda pada bagian data yang berbeda (Flynn, 1966). Secara umum, ada dua tipe arsitektur MIMD, yaitu *true multiprocessor* dan *shared resource multiprocessor*. *Lockout time* atau waktu tunda prosesor dalam mengakses informasi dirumuskan sebagai berikut (Madnick, 2011).

$$L_j = \sum_i p_{ij} T_{ij}$$

B. Contoh Aplikasi Komputasi Paralel di *Shared Memory*

Studi Kasus 1

Salah satu aplikasi *shared memory* adalah masalah OpenMP, misalnya dalam GAMESS (Bak et. al., 2021) GAMESS (*(General Atomic and Molecular Electronic Structure Sys-*

tem) adalah paket perangkat lunak dengan berbagai struktur elektronik metode kimia kuantum, seperti Hartree–Fock (HF) dan teori gangguan Moller-Plesset orde kedua dengan resolusi pendekatan identitas. Sebagian besar kodennya di Fortran 77/90, dengan pustaka C/C++ opsional yang menggunakan CUDA untuk membongkar kode ke GPU NVIDIA. GAMESS secara tradisional menggunakan MPI bersama dengan OpenMP untuk dijalankan pada CPU multi-core. Beberapa metode di GAMESS telah diperbarui untuk menggunakan OpenMP secara opsional untuk membongkar wilayah yang secara komputasi mengarah ke GPU. Penelitian Bak (2021) difokuskan pada port GPU metode HF dan RI-MP2 menggunakan OpenMP.

Adapun alur GAMESS adalah sebagai berikut. Metode HF memecahkan satu set persamaan nilai eigen non-linier iteratif untuk energi sistem molekuler. Ini memiliki dua langkah utama, yaitu: i) perhitungan sejumlah besar (pada urutan 4 yang merupakan ukuran sistem molekuler) dari 4-indeks 2-elektron integral (4-2ERI); dan ii) membentuk matriks 2 Fock dengan mengontraksi tensor 4 4-2ERI dengan matriks densitas. HF menggunakan metode dasar yang merupakan titik awal untuk banyak metode dengan akurasi yang lebih tinggi seperti metode RI-MP2. Dalam metode RI-MP2, 4-2ERI didekati sebagai produk dari 3-2ERI. Ini menyederhanakan integral evaluasi dari integral tolakan 2-elektron 4-indeks ke 3-indeks dan memungkinkan penggunaan operasi perkalian matriks yang efisien.

Selanjutnya, implementasi dan pengoptimalan dengan OpenMP adalah sebagai berikut. Untuk kode HF, penelitian Bak (2021) berfokus pada evaluasi 4-2ERI. Paralel dengan threading MPI+OpenMP pada CPU, kode mempertahankan beberapa tingkat pernyataan kondisional tambahan, karena pekerjaan komputasi tidak ditugaskan secara

merata ke atas, akan timbul kesulitan dari ketidakseimbangan beban. Untuk mengatasinya, direorganisasi aliran kontrol dan urutan integral dihitung dengan menempatkan kondisional ke dalam blok kode terpisah dan menyortir integral sebelumnya. Hanya beberapa arahan OpenMP yang ditambahkan ke kode yang diatur ulang. Arahan OpenMP dimasukkan ke kode offload untuk GPU (dan subrutin yang dipanggil dari wilayah target dianotasi dengan

'menyatakan target'). Diperhatikan bahwa rutinitas untuk menghitung integral, misalnya, int1, tidak dimodifikasi sama sekali. Versi GAMESS ini sedang dalam pengembangan cabang dan mendukung satu jenis integral. Untuk kode RI-MP2, difokuskan pada perhitungan koreksi perturbatif, yaitu didominasi oleh panggilan ke rutinitas perkalian matriks (DGEMM). Di sini, strategi untuk port dari CPU ke GPU adalah untuk menggabungkan bagian tertentu sehingga input ke panggilan DGEMM lebih besar, menghasilkan lebih tinggi intensitas aritmatika per panggilan DGEMM, dan peluncuran kernel yang lebih sedikit.

Studi Kasus 2

Bak (2021) juga membahas tentang GenASiS (*General Astrophysical Simulation System*). GenASiS adalah kode yang dikembangkan untuk simulasi skala besar fenomena astrofisika yang menargetkan superkomputer. Saat ini, ini GenASiS ditujukan untuk mensimulasikan dan memodelkan supernova core-collapse dan fenomena terkait peristiwa supernova yang dapat diamati. Simulasi menggunakan GenASiS telah menyebabkan penemuan amplifikasi medan magnet oleh Stasioner Accretion Shock Instability (SASI) di lingkungan supernova.

Implementasi GenASiS menggunakan OpenMP adalah sebagai berikut. Inti dari fasilitas penyimpanan data di GenASiS Basics adalah Kelas StorageForm, yang terdiri dari anggota dan metode untuk menyimpan data generik dan metadata. Data disimpan sebagai dua dimensi array di mana indeks pertama biasanya menyebutkan sel-sel di mesh dan indeks kedua menyebutkan variabel. Metadata termasuk unit dan nama variabel yang dapat digunakan untuk I/O dan visualisasi. Sebagian besar pemecah dan kebutuhan penyimpanan GenASiS ditulis menggunakan kelas StorageForm, memungkinkan kode yang seragam dan disederhanakan untuk fungsionalitas seperti I/O dan pertukaran sel hantu tetangga terdekat.

Implementasi OpenMP dari GenASiS memperluas Kelas StorageForm dengan metode untuk mencerminkan, mengaitkan, dan menyinkronkan alokasi data memori

host (CPU) dengan alokasi memori perangkat (GPU) yang sesuai. Gambar 1 mengilustrasikan penggunaan kelas StorageForm dengan metode yang relevan untuk dikelola alokasi dan asosiasi memori perangkat. Pada Gambar 1, baris 4 mengalokasikan anggota kelas array dua dimensi nilai pada memori host. Baris 5 mencerminkan alokasi tersebut pada perangkat dan mengaitkan lokasi memori di host dengan yang di perangkat. Ini dilakukan dengan menggunakan runtime perpustakaan OpenMP rutin `omp_target_associate_ptr()` di bawah tenda. Dengan asosiasi ini, runtime OpenMP bertemu dengan host itu variabel dalam wilayah target, pemetaan implisit, dan transfer data dihindari karena pemetaan tersebut sudah ada. Baris 7 menginisialisasi nilai anggota dengan kondisi khusus masalah. Baris 8 memperbarui salinan perangkat variabel dengan nilai awal, yang kemudian dapat digunakan di dalam rutin `AddKernel()` pada perangkat.

```
1 type ( StorageForm ) :: &
2     Fields
3 ...
4 call Fields % Initialize ( [ nCells , nVariables ], ... )
5 call Fields % AllocateDevice ( )
6 ...
7 call SetInitialConditions ( Fields % Value )
8 call Fields % UpdateDevice ( )
9 call AddKernel &
10    ( Fields % Value ( :, 1 ), &
11      Fields % Value ( :, 2 ), &
12      Fields % Value ( :, 3 ),
13      UseDevice = .true. )
14 call Fields % UpdateHost ( )
```

Gambar 1. Kode StorageForm untuk GenASIS

Daftar Pustaka

- Azis, Mohammad T, Azizul K, & Bintoro A. 2016. Kinerja Perhitungan Kritikalitas Mcnp6 Pada Komputasi Paralel. *Jurnal Seminar Keselamatan Nuklir*: diakses dari https://inis.iaea.org/collection/NCLCollectionStore/_Public/50/022/50022722.pdf tanggal 13 Oktober 2022
- Akbar, Auriza R. 2014. *Pemrosesan Paralel*. diakses dari <https://adoc.pub/pemrosesan-paralel-auriza-rahmad-akbar.html> tanggal 12 Oktober 2022
- Annas, Mayzar., I Gede, L.A. Syamsul Irfan. 2010. Studi Komputasi Paralel dan Implementasinya pada Kasus Komputasi Matriks Besar. diakses dari http://jcosine.if.unram.ac.id/public/journals/1/jCossin2016_PaperExp.pdf pada 12 oktober 2020
- Arvin, dkk. 2017. Optimisasi Pustaka Untuk Perkalian Matriks Menggunakan Algoritma Strassen Berbasis Opencl. *Jurnal Seminar Nasional Inovasi Dan Aplikasi Teknologi Di Industri 2017*. diakses dari <https://ejournal.itn.ac.id/index.php/seniasi/article/view/1591/141> 5 tanggal 11 Oktober 2022
- Bak, Seonmyeong, et. al. 2021. *OpenMP Application Experiences: Porting to Accelerating Nodes*. *Parallel Computing* 109 (2021).

Flynn, Michael J., Rudd, Kevin W. 1996. *Parallel Architectures*. CRC Press.

Flynn, Michael J. (December 1966). "Very high-speed computing systems". *Proceedings of the IEEE*. **54** (12): 1901–1909

Kurniawan, A. 2010. Pemrograman Paralel dengan MPI & C. ANDI Yogyakarta

Kurniawan, dkk. 2015. Analisis Perbandingan Komputasi GPU dengan CUDA dan Komputasi CPU untuk Image dan Video Processing. *Jurnal Seminar Nasional Aplikasi Teknologi Informasi (SNATi)*. diakses dari <http://etd.repository.ugm.ac.id/pemelitian/detail/117016> tanggal 11 Oktober 2022

Madnick, S.E. "Multi-processor software lockout," in Proc. concerning parallel processing and parallel processors," Proc. 1968 Ass. Comput. Mach. Nat. Conf., pp. 19-24.

Mohr, dkk. 2001. Design and Prototype of a Performance Tool Interface for OpenMP. *Journal of Supercomputing*. diakses dari https://www.researchgate.net/publication/2380669_Design_and_Prototype_of_a_Performance_Tool_Interface_for_OpenMP tanggal 11 Oktober 2022

Nudirman, Dani. 2016. Apa itu CUDA, GPU Acceleration, dan Mercury Playback Engine? diakses dari rumaheditor.com: <http://rumaheditor.com/apa-itu-cuda-gpu-acceleration-dan-mercury-playback-engine/> tanggal 11 Oktober 2022

Rahmatullah dkk. 2021. Analisis Perbandingan Performa Pemrograman Sekuensial dan Paralel Dengan Skema Uji Matrix, Filter Dan Quick Sort. *Jurnal Teknologi Informasi Universitas Lambung Mangkurat (JTIULM)* 6. 19 - 24. 10.20527/jtiulm.v6i1.69. diakses dari https://www.researchgate.net/publication/351300511_Analisis_Perbandingan_Performa_Pemrograman_Sekuensial_Dan_Paralel_Dengan_Skema_Uji_Matrix_Filter_Dan_Quick_Sort tanggal 11 Oktober 2022

Rosocha, W. G. & Lee E.S. 1979. Performance Enchacement of SISD Processors. *Journal of Computer Systems Research Group, University of Toronto, Canada.* diakses dari <https://dl.acm.org/doi/pdf/10.1145/800090.802912> tanggal 13 Oktober 2022

Schmidt, Bertil, et. Al. 2018. *Parallel Programming: Concept and Practice*. Morgan Kaufmann.

Stokes, Jon. 2000. *SIMD architecture*. diakses dari <https://arstechnica.com/features/2000/03/simd/> tanggal 13 Oktober 2022

Suhartanto, Heru. 2006. Kajian Perangkat Bantu Komputasi Paralel pada Jaringan PC. Makara, Teknologi, Vol. 10, no.2 November 2006. Diakses pada 12 oktober 2022 di <https://media.neliti.com/media/publications/148472-ID-kajian-perangkatbantu-komputasi-parallel.pdf>

Torres, Jordi. *Deep Learning Frameworks for Parallel and Distribution Structure*. Diakses pada tanggal 28 Oktober 2022 di <https://towardsdatascience.com/scalable-deep-learning-on-parallel-and-distributed-infrastructures-e5fb4a956bef>.

Quinn, Michael J. *Parallel Programming in C with MPI and OpenMP*. Boston: McGraw Hill, 2004.

Wanasurya, Andri L, Sri M, & Maria A K. 2017. Komputasi Paralel Untuk Pengolahan Prestasi Akademik Mahasiswa. *Jurnal Teknologi Elektro, Universitas Mercu Buana*. diakses dari <https://publikasi.mercubuana.ac.id/index.php/jte/article/view/2180> tanggal 13 Oktober 2022

Wibawa, dkk. 2018. Komputasi Paralel Menggunakan Model Message Passing Pada SIM RS (Sistem Informasi Manajemen Rumah Sakit). *Jurnal Majalah Ilmiah Teknologi Elektro, Vol. 17, No. 3, September - Desember 2018.* diakses dari <https://pdfs.semanticscolar.org/f12>

Review Article

Pengolahan Citra Digital Pada Komputasi Paralel

Dalam Mendeteksi Kebakaran (API)

Eka Dwi Agustina Ginting

ABSTRAK

Kebakaran adalah kejadian timbulnya api yang tidak terkendali yang dapat membahayakan jiwa maupun harta benda. Kebakaran sulit untuk diprediksi kapan dan dimana terjadinya. Misalnya kebakaran hutan, kebakaran pada perumahan atau bangunan bisa terjadi pada siang maupun malam hari. Kebakaran menyebabkan kerugian yang sangat besar seperti ratusan bahkan juta rumah dan bangunan serta jutaan hektar hutan musnah akibat kebakaran setiap tahun. Kebakaran terus mengancam sistem ekologi, infrastruktur, dan keselamatan publik. Pengolahan citra digital menggunakan pendekatan pengolahan citra berwarna yang dapat memberikan lokasi hotspot sehingga analisis dan tindakan dapat dilakukan dengan cepat dan akurat. Dengan deteksi dini diharapkan dampak kebakaran dapat diatasi dengan cepat, tepat, dan akurat. Manfaat yang akan diperoleh adalah penanganan kebakara agar tidak meluas dan menimbulkan gangguan.

Kata kunci: kebakaran, Pengelolaan Citra, deteksi, GPU.

1. PENDAHULUAN

1.1 Latar Belakang

Kebakaran adalah kejadian timbulnya api yang tidak terkendali yang dapat membahayakan jiwa maupun harta benda. Kebakaran sulit untuk diprediksi kapan dan dimana terjadinya. Misalnya kebakaran hutan, kebakaran pada perumahan atau bangunan bisa terjadi pada siang maupun malam hari. Karena sulit untuk memperkirakan lokasi dan waktu terjadinya kebakaran maka fungsi kamera CCTV yang sudah banyak dipasang pada bangunan dan tempat-tempat umum sangat efektif untuk memantau lokasi setiap waktu. Dengan ini dampak dari kebakaran dapat diminimalisir dengan adanya deteksi secara dini [2].

Deteksi api merupakan salah satu teknologi yang saat ini dikembangkan seiring perkembangan teknologi yang sangat cepat dan diikuti dengan perkembangan di bidang Artificial Intelligence (AI) atau Kecerdasan Buatan. Salah satu cabang AI salah satunya yaitu computer vision. Computer vision tentunya tidak lepas dari penggunaan library OpenCV. OpenCV merupakan library opensource yang tujuannya dikhusruskan untuk melakukan pengolahan citra. Penggunaan computer vision dalam alat pendekripsi kebakaran sangat penting, karena dengan menggunakan computer vision petugas akan mengetahui keadaan ruangan atau gedung yang terbakar [3]

Banyak kebakaran hutan yang tidak dapat ditanggulangi secara dini dikarenakan letak lokasi titik api sulit untuk diketahui dengan cepat. Lokasi titik api yang sulit dijangkau oleh petugas pemadam kebakaran hutan merupakan salah satu faktor yang mempengaruhi perluasan kebakaran hutan disamping faktor-faktor lainnya. Titik api yang merupakan masalah dari kebakaran hutan memang menjadi konsentrasi utama bagi pemadam kebakaran hutan, karena jika pengendalian awal pada titik api dapat dilakukan maka kebakaran hutan diharapkan tidak meluas dan gangguan dari dampak kebakaran hutan tidak akan menimbulkan masalah yang baru. Kebakaran hutan dan lahan diidentifikasi dari sebaran asap yang dihasilkan pada olahan citra dan dikonfirmasi dengan titik panas dari citra yang menggunakan pola spektral untuk mendekripsi kebakaran hutan dan lahan.

Oleh karena itu, dibangun system yang dapat mendekripsi kebakaran pada video berbasis pengolahan citra dengan dukungan GPU yaitu deteksi kebakaran secara real time, pendekripsi api dengan computer vision dan kebakaran hutan dengan Ekstraksi Exif Pada Informasi Gambar dan citra satelit Himawari-8.

2. TINJAUAN PUSTAKA

2.1 Pemrograman Paralel

Pemrosesan paralel merupakan teknik komputasi menggunakan dua atau lebih komputer untuk menyelesaikan suatu tugas dalam waktu yang simultan.

2.2 Pengolahan Citra

Citra digital adalah biner dari citra dua dimensi berupa elemen berbentuk array yang disebut piksel yang tentunya memiliki nilai numerik. Citra dalam perwujutan dapat bermacam-macam, mulai dari gambar putih pada sebuah foto (yang tidak bergerak).

2.3 Deteksi Gerakan dengan Adaptive-Gaussian Mixture Model (Adaptive-GMM)

Rancangan sistem deteksi kebakaran pada video berbasis pengolahan citra dengan dukungan GPU terdiri dari pembacaan data frame video, inisialisasi model untuk Adaptive-GMM, membuat tabel warna, praproses frame video, deteksi gerakan dengan Adaptive-GMM, segmentasi warna api, mendeteksi kebakaran dari kombinasi hasil deteksi gerakan dan segmentasi warna api, dan menampilkan hasil deteksi pada layar. GPU digunakan pada metode yang memerlukan komputasi yang tinggi seperti konversi warna ke ruang warna grayscale dan HSV, deteksi gerakan dan segmentasi warna api [2].

Deteksi gerakan menggunakan metode Adaptive-Gaussian Mixture Model. Metode ini mendekripsi gerakan dengan memodelkan setiap piksel ke dalam M komponen distribusi Gaussian serta melakukan perbaikan bobot (w), rata-rata (μ) dan standar deviasi (σ) pada masing-masing komponen Gaussian tersebut. Kemungkinan sebuah piksel mempunyai nilai \rightarrow pada waktu T ditunjukkan pada persamaan (1) [2].

dimana adalah parameter bobot, adalah perkiraan rata-rata dan adalah perkiraan varian pada komponen Gaussian ke- m . Bobot tidak negatif dan jumlah dari total bobot selalu satu sehingga diperlukan normalisasi setelah perbaikan bobot. Bila ada data baru pada waktu t perbaikan model untuk bobot, rata-rata dan varian ditunjukkan pada persamaan (2), (3) dan (4) secara berurutan. [2].

dimana dan kira-kira . Untuk data baru dengan jarak komponen terdekat dan bobot terbesar, keanggotaan bernilai 1 selain itu bernilai 0. Jarak terdekat apabila jarak mahalanobisnya kurang dari tiga standar deviasi. Jarak kuadrat dari komponen ke-m yaitu B . Bila tidak ada komponen dengan jarak terdekat maka dibuat komponen baru dengan dimana adalah inisial varian. Bila jumlah maksimum komponen tercapai, komponen dengan bobot terendah dapat dibuang. Komponen diurutkan berdasarkan bobotnya dari besar ke kecil sehingga latar belakang (B) dapat dihitung dengan persamaan (5) [2].

dimana adalah maksimum bagian dari data yang boleh masuk dalam obyek bergerak tanpa mempengaruhi model latar belakang. Pemilihan jumlah komponen dilakukan secara adaptif tapi dengan batasan maksimum jumlah komponen yang diperbolehkan. Oleh karena itu, perbaikan bobot pada persamaan (2) perlu diubah menjadi persamaan (6) [2].

dimana . adalah bias dengan c adalah koefisien dan T adalah jumlah sampel pada dataset. Kumulatif bobot dijaga agar selalu satu dan bila ada komponen dengan bobot negatif maka komponen tersebut dapat dibuang [2].

2.4 You Only Look Once (YOLO)

You Only Look Once (YOLO) merupakan object detection network yang dibuat oleh Joseph Redmon di tahun 2016. Cara kerja YOLO cukup sederhana, YOLO menerima sebuah input image yang dibagi menjadi grid sebesar $S \times S$ yang dikirimkan ke sebuah neural network untuk membuat bounding box dan class prediction. Setiap grid cell memprediksi B bounding box dan confidence score dari tiap kotak. Confidence score merupakan nilai probabilitas yang merefleksikan seberapa tingkat kepercayaan model bahwa objek di dalam kotak berupa objek yang diprediksikan [5].

2.5 Deteksi Api Kebakaran dengan Metode YOLO.

Metode yang digunakan untuk mendeteksi api kebakaran dengan menggunakan metode YOLOv3-tiny dengan menggunakan pre-trained weight dan model dari darknet open source YOLO. Sehingga perlu menyesuaikan kebutuhan software dan hardware dalam mengimplementasikan YOLOv3-tiny dari darknet untuk dataset yang digunakan dalam penelitian. Proses uji coba dilakukan dengan melakukan proses pengujian metode dengan menguji beberapa kondisi. Proses uji coba dilakukan untuk mengetahui batasan deteksi api. Sedangkan proses evaluasi dilakukan untuk mengetahui tingkat keakuratan kinerja model

dan untuk menganalisis hasil uji coba validasi. Validasi kinerja program dilakukan dengan menghitung seberapa besar nilai akurasi klasifikasi objek [3].

2.6 Algoritma Haar Cascade Classifier

Algoritma Haar Cascade Classifier merupakan sebuah metode untuk mendeteksi objek dalam sebuah gambar, video, dan secara real-time. Metode ini ditemukan oleh Paul Viola dan Michael Jones pada tahun 2001, metode ini juga adalah gabungan dari fungsi Haar-Like dan cascade classifier yang bertujuan untuk membentuk klasifikasi. Fungsi Haar-like-feature atau biasa disebut juga dengan Haar cascade classifier adalah fungsi persegi panjang (persegi) yang memberikan indikasi spesifik pada gambar. Pengklasifikasi Haar Cascade berasal dari gabungan piksel hitam dan piksel putih yang membentuk kotak [5].

3. METODE

Penelitian ini merupakan penelitian sekunder berjenis literature review atau review artikel yang berarti analisis berupa kritik yang (baik kritik yang membangun ataupun kritik yang menjatuhkan) dari penelitian yang telah dilakukan terhadap suatu topik khusus atau pertanyaan terhadap suatu bagian dari keilmuan tertentu. Penulisan literature review ini berdasarkan jurnal yang dikumpulkan melalui situs internet seperti google, google cendekia (scholar), researchgate, science direct, IEEE, dan situs e-journal universitas di Indonesia lainnya. Artikel ilmiah dikumpulkan dengan mencari di situs web tersebut dengan kata kunci “Deteksi kebakaran dengan pengolahan citra”, “pendeteksi kebakaran berbasis GPU”, dan sebagainya.

4. HASIL DAN PEMBAHASAN

4.1 Hasil

Hasil dari analisis terhadap jurnal-jurnal yang digunakan dalam literatur review ini dapat dilihat pada table berikut.

NO	Judul dan Penulis	Kesimpulan
1	Aplikasi Pengolahan Citra Sebagai Pendekripsi Dini Kebakaran Menggunakan Colour Image Processing (Atthariq dkk, 2017)	Hasil rata-rata akurasi system yang didapatkan dari deteksi kebakaran pada siang hari yaitu 97,96% dengan kesalahan deteksi sebesar 0,25% dan akurasi sebesar 98,65% dengan kesalahan deteksi sebesar 0,85% untuk deteksi kebakaran pada malam hari dari 3 video dengan 5 kali percobaan.
2	Deteksi Kebakaran pada Video Berbasis Pengolahan Citra dengan Dukungan GPU. (Prahara, 2015)	Hasil rata-rata akurasi system deteksi kebakaran dengan 15 sampel video pada siang hari yaitu 97,96 % dengan kesalahan rata-rata sebesar 0.25% dan akurasi rata-rata sebesar 98,65% dengan kesalahan rata-rata sebesar 0.85% pada malam hari.
3	Deteksi Api Kebakaran Berbasis Computer Vision dengan Algoritma YOLO (Widharma dkk , 2022)	nilai accuracy sebesar 0.8, precision sebesar 1 dan recall 0.8 pada siang hari. Sedangkan pada malam hari mendapatkan hasil accuracy sebesar 0.96, precision sebesar 1 dan recall 0.96.

4	Deteksi Kebakaran Hutan dan Lahan Menggunakan Citra Satelit Himawari-8 di Kalimantan Tengah. (Sepriando, A., Hartono., Jatmiko, R.H. 2019).	Citra himawari-8 dapat diolah menjadi titik panas untuk identifikasi kebakaran hutan dan lahan dengan resolusi spasial dan resolusi temporal 10 menit.
5	Sistem Deteksi Api Menggunakan Pengolahan Citra Pada Webcam Dengan Metode Yolov3 (Ramadah dkk, 2022)	Nilai rata-rata akurasi sebesar 91,60% dan nilai rata-rata presisi sebesar 83,73% dan dengan menggunakan metode Haar Cascade Classifier pengujian pendekripsi pada objek berwarna dengan beberapa background yang dipengaruhi jarak dan intensitas cahaya didapatkan nilai rata-rata akurasi sebesar 61,93% dan nilai rata-rata presisi sebesar 30,26%.
6	Deteksi Dini Kebakaran Hutan dan Lahan Memanfaatkan Ekstraksi Exif Pada Informasi Gambar Berbasis Pengolahan Citra (Wibowo dkk, 2021)	Metadata pada lokasi terkadang masih gagal didapatkan, Nilai accuracy klasifikasi api sebesar 75% dengan precision sebesar 80% dan recall sebesar 80%. Hasil untuk nilai accuracy klasifikasi asap sebesar 70% dengan precision sebesar 92% dan recall sebesar 87% dan hasil dari pengolahan citra masih terdapat false detection karena banyak pixel yang bukan asap atau api ikut terklasifikasi.

4.2 Pembahasan

Penelitian [1] dan [2] Aplikasi pengolahan citra sebagai pendekripsi dini kebakaran menggunakan webcam atau CCTV sebagai alat untuk mendekripsi api. Dalam penelitian [1] dan [2] dibangun sistem yang dapat mendekripsi kebakaran pada video berbasis pengolahan citra dengan dukungan GPU (*Graphic Processing Unit*) dengan penerapan deteksi gerak dan

segmentasi api menggunakan metode CIP (*Colour Image Processing*) untuk penelitian [1] dan penelitian [2] menggunakan metode *Adaptive-Gaussian Mixture Model (Adaptive-GMM)* sehingga tercapai system deteksi kebakaran secara real time.

Pada penelitian [1] dan [2] Kecepatan diukur dalam milidetik dan jumlah frame yang dieksekusi per detik (fps). Pada setiap kategori resolusi digunakan 3 video dengan fps 30 berdurasi rata-rata 10 detik untuk diukur kecepatan eksekusi metodenya selama 5 kali percobaan (15 sampel video). System akan memproses data video dengan menerapkan hasil segmentasi warna api untuk mendeteksi kebakaran. Pengujian akurasi dan kesalahan system dalam mendeteksi kebakaran dilakukan pada sampel video beresolusi 640x480 dengan 3 kelompok berdasarkan jumlah video kebakaran. akurasi rata-rata yang didapatkan dari deteksi kebakaran pada siang hari yaitu 97,96% dengan kesalahan deteksi sebesar 0,25% dan akurasi sebesar 98,65% dengan kesalahan deteksi sebesar 0,85% untuk deteksi kebakaran pada malam hari. Sistem mampu mendeteksi kebakaran secara real time yaitu sekitar 35,46 fps pada resolusi video 800x600 dan lebih cepat pada resolusi yang lebih kecil. Kesalahan deteksi terutama disebabkan adanya perubahan cahaya yang mendadak dari sumber api terhadap lingkungan sekitar dan pantulan cahaya kebakaran pada permukaan yang mengkilap.

Penelitian [3] pengolahan citra sebagai pendeksi kebakaran dan mengetahui kondisi atau keadaan saat terjadi kebakaran menggunakan kamera, maka dibangun system yang dapat mendeksi api kebakaran berbasis computer vision dengan menggunakan algoritma *You Only Look Once (YOLO)* untuk mendeksi api kebakaran serta disertai dengan notifikasi telegram dan buzzer. Pada pengujian deteksi api menggunakan 4 sumber api yang berbeda yaitu lilin obor kecil, obor besar dan sabut kelapa. Dari data pengujian deteksi api di berbagai background mendapatkan hasil nilai accuracy sebesar 0.8, precision sebesar 1 dan recall 0.8 pada siang hari. Sedangkan pada malam hari mendapatkan hasil accuracy sebesar 0.96, precision sebesar 1 dan recall 0.96. Tingkat nilai accuracy dan recall pada pengujian berbagai jarak menghasilkan sistem deteksi api akan mengalami penurunan seiring jauhnya jarak api dengan kamera. Untuk nilai precision menghasilkan nilai 1 di berbagai jarak pada siang dan malam hari. Hal ini menunjukkan ketepatan hasil klasifikasi sebesar 100 %. Nilai precision yang stabil tersebut dipengaruhi oleh pembacaan deteksi api, dimana sistem tidak pernah mendeksi objek lain sebagai api.

Penelitian [4] pengolahan citra dengan memanfaatkan data citra satelit Himawari-8 untuk deteksi kebakaran hutan dan lahan yang menghasilkan titik panas dengan resolusi temporal 10 menit, dimana hasilnya di validasi dengan citra polar dan data kebakaran lapangan. Lokasi penelitian berada di Provinsi Kalimantan Tengah dan waktu penelitian adalah bulan September 2019. Data yang digunakan untuk pengolahan adalah 5 saluran AHI, peta batas administrasi dan tutupan lahan. Pemrosesan data citra satelit mencakup pemilihan piksel penutup lahan dan batas administrasi, penentuan waktu pengamatan, eliminasi piksel awan, Algoritma Pemantau Kebakaran Aktif, dan validasi hasil. Data citra Himawari-8 dapat diolah menjadi titik panas dengan temporal 10 menit. Validasi terhadap citra polar memiliki tingkat akurasi 66,2%-75,4%, comission error 28,2-46,9% dan omission error 24,6-33,8%. Tingginya comision error terhadap citra VIIRS dikarenakan citra VIIRS memiliki resolusi spasial yang jauh lebih tinggi dibandingkan dengan citra Himawari-8. Validasi titik panas Himawari-8 dengan data kebakaran lapangan menghasilkan akurasi 31,6% dan omission error 68,4%. Rendahnya nilai akurasi karena koordinat kebakaran lapangan berada di area perbatasan area yang terbakar dan tidak terbakar.

Penelitian [5] pengolahan citra dengan sistem deteksi api pada webcam menggunakan pengolahan citra yaitu metode YOLOv3 yang akan dibandingkan dengan metode Haar Cascade classifier dalam hal akurasi dan presisi. Pengujian pendekripsi pada objek berwarna dilakukan dengan beberapa background yang dipengaruhi jarak dan intensitas cahaya menggunakan metode YOLOv3. Pengujian dilakukan bertujuan untuk mengetahui seberapa akurat dan presisi sistem mendekripsi sebuah objek yang dipengaruhi warna background, jarak, dan intensitas cahaya sehingga pada pengujian menggunakan metode YOLOv3 didapatkan nilai rata-rata akurasi sebesar 91,60% dan nilai rata-rata presisi sebesar 83,73% dan dengan menggunakan metode Haar Cascade Classifier pengujian pendekripsi pada objek berwarna dengan beberapa background yang dipengaruhi jarak dan intensitas cahaya didapatkan nilai rata-rata akurasi sebesar 61,93% dan nilai rata-rata presisi sebesar 30,26%.

Penelitian [6] pengolahan citra dengan aplikasi android diusulkan untuk mengekstrak metadata foto Exchangeable Image Format (EXIF). Metadata tersebut memiliki informasi citra seperti lintang dan bujur, untuk mendapatkan lokasi kebakaran hutan yang dilaporkan oleh pengguna aplikasi. Penelitian ini menggunakan data yang didapatkan dari google gambar, dari Pusat Pengendalian Operasi Pengendalian Bencana (Pusdalops PB) BPBD Pulang Pisau serta pengambilan gambar secara langsung menggunakan kamera ponsel.

Data yang dikumpulkan yaitu data latih dan data uji. Data latih berjumlah 25 data citra asap dan 25 data citra api serta data uji sebanyak 20 data citra asap dan 20 data citra api. Metode penyaringan warna yang digunakan berdasarkan pada ruang warna Red Green Blue (RGB), Hue Saturation Value (HSV) dan YcbCr dan untuk klasifikasi gambar api, peneliti mengonversi gambar RGB menjadi salah satu ruang warna yaitu YCbCr. Sedangkan pada klasifikasi asap peneliti mengonversi RGB ke HSV dan Grayscale. Hasil dari proses klasifikasi kebakaran dan asap dideskripsikan dengan menggunakan confusion matrix.

Confusion matrix dapat memberikan informasi perbandingan hasil klasifikasi yang dilakukan oleh sistem (model) dengan hasil klasifikasi sebenarnya sehingga pada penelitian ini confusion matrix menghasilkan tingkat akurasi 75%, presisi 80% dan recall 80% untuk klasifikasi kebakaran dan akurasi 70%, presisi 92% dan recall 87% untuk klasifikasi asap. Ada 25% dan 30% salah klasifikasi data kebakaran dan asap. Hal ini dikarenakan metode color filtering mengelompokkan setiap piksel warna dari citra, oleh karena itu banyak piksel yang tidak diklasifikasikan sebagai citra api atau asap yang diklasifikasikan karena terdapat objek lain yang memiliki rentang warna untuk mengklasifikasikan api dan asap.

5. KESIMPULAN

Berdasarkan hasil studi deteksi kebakaran dengan pengolahan citra, dapat disimpulkan bahwa tingkat persentase akurasi klasifikasi dipengaruhi dengan data train (data latih). Selain itu konversi dari RGB ke citra lain dipengaruhi oleh fungsi konversinya. Meskipun sistem deteksi kebakaran dengan pengolahan citra sudah diterapkan, penggunaan admin sebagai validasi kebakaran akan sangat mempengaruhi hasil dari penerapan sistem ini.

DAFTAR PUSTAKA

- [1] Athariq., Nasir, M., Azhar., Hendrawaty. (2017). Aplikasi Pengolahan Citra Sebagai Pendeksi Dini Kebakaran Menggunakan Colour Image Prosessing. *Proceeding Seminar Nasional Politeknik Negeri Lhokseumawe: Vol.1 No.1.*
- [2] Prahara, Adhi. 2015. Deteksi Kebakaran pada Video Berbasis Pengolahan Citra dengan Dukungan GPU.
- [3] Widharma, I.G.S., Sntiary, P.A.W., Sunaya, I.N., Darminta, I.K., Sangka, I.G.N., Widiatmika, P.A.W. (2022). Deteksi Api Kebakaran Berbasis Computer Vision Dengan Algoritma YOLO. *Journal of Applied Mechanical Engineering and Green Technology*.
- [4] Sepriando, A., Hartono., Jatmiko, R.H. (2019). Deteksi Kebakaran Hutan dan Lahan Menggunakan Citra Satelit Himawari-8 di Kalimantan Tengah. *Jurnal Sains & TEknologi Modifikasi Cuaca, Vol.20, No.2, PP. 79-89.*
- [5] Ramadah, F., Wibawa, IG.P.D., Rizal, A. (2022). Sistem Deteksi Api Menggunakan Pengolahan Citra Pada Webcam Dengan Metode Yolov3. *e-Proceeding of Engineering: Vol.9, No.2 Page 226.*
- [6] Wibowo, R.E., Teguh, R., Lestari, A., (2021). Deteksi Dini Kebakaran Hutan Dan Lahan Memanfaatkan Ekstraksi Exif Pada Informasi Gambar Berbasis Pengolahan Citra. *Jurnal Teknologi Informasi: Vol 15 No.1.*