Modul 7

Mpi4py

A. Tujuan Praktikum

1. Memahami konsep dasar *Message Passing Interface (MPI)* pada sistem memori terdistribusi.
2. Menjelaskan mekanisme komunikasi antar proses (*send* dan *receive*) dalam paradigma *message passing*.
3. Menerapkan komunikasi antar node menggunakan Python dengan pustaka MPI4PY.
4. Mengidentifikasi efisiensi dan kelebihan model *distributed memory* dibanding *shared memory*.

B. Materi

1. Konsep Dasar *Distributed Memory*

   Menurut Richard M. Fujimoto (2000) dalam *Parallel and Distributed Simulation Systems*, sistem *distributed memory* merupakan arsitektur di mana setiap prosesor memiliki memori lokal sendiri. Komunikasi antar prosesor dilakukan dengan pertukaran pesan (message passing). Berbeda dengan *shared memory* yang menggunakan ruang memori bersama, *distributed memory* menuntut koordinasi eksplisit melalui pesan antar node. Dalam *Distributed Memory*:

   - Skalabilitas tinggi (dapat diperluas dengan mudah ke banyak node).
   - Cocok untuk sistem komputasi klaster dan simulasi besar.
   - Tidak terjadi *memory contention*.
   - Komunikasi lebih kompleks karena memerlukan sinkronisasi eksplisit.
   - Overhead komunikasi bisa meningkat saat jumlah node bertambah.

2. Message Passing Interface (MPI)

   Menurut Fujimoto (2000), Message Passing Interface (MPI) adalah standar komunikasi antar proses pada sistem paralel dan terdistribusi, yang digunakan untuk memungkinkan program yang berjalan di beberapa prosesor atau komputer berkoordinasi melalui pertukaran pesan (message passing). MPI bukan bahasa pemrograman, tetapi kumpulan fungsi, protokol, dan pustaka yang dapat digunakan oleh berbagai bahasa (C, C++, Fortran, Python) untuk menjalankan komputasi paralel di sistem distributed memory. Menurut Taniar et al. (2008) dalam *High-performance Parallel Database Processing and Grid Databases*, MPI adalah standar industri untuk melakukan komunikasi antar proses dalam sistem paralel. Tujuan dari MPI adalah :

   - Menyediakan mekanisme komunikasi efisien antar proses.
   - Memungkinkan portabilitas (bisa dijalankan di berbagai sistem paralel).
   - Memastikan determinisme komunikasi (pesan diterima sesuai urutan pengiriman).
   - Mendukung skala besar (dari dua proses hingga ribuan node HPC).

Sesuai penjelasan Taniar et al. (2008), MPI bekerja dalam ruang komunikasi yang disebut communicator. Contoh analoginya adalah sebuah grup WhatsApp (communicator). Setiap anggota punya nomor unik (rank). Pesan yang dikirim di grup hanya dapat dibaca oleh anggota dalam grup tersebut. Komponen utama dari MPI adalah:

| Komponen | Deskripsi |
|---|---|
| Communicator | Ruang kerja di mana proses dapat berkomunikasi (default: MPI.COMM_WORLD). |
| Rank | Nomor unik yang mengidentifikasi setiap proses di dalam communicator (0, 1, 2, ...). |
| Size | Jumlah total proses di dalam communicator. |

Menurut Fujimoto, setiap proses MPI berjalan secara independen dengan memori sendiri (tidak berbagi RAM), berinteraksi dengan proses lain hanya melalui pengiriman pesan dan tidak mengetahui isi memori proses lain. Maka, setiap proses dapat dianggap sebagai node otonom dalam jaringan yang berkolaborasi menyelesaikan satu tugas besar. Jenis Komunikasi dalam MPI

a. Komunikasi Point-to-Point

   Menurut Davies (2019), komunikasi dalam jaringan selalu melibatkan dua pihak utama: pengirim (sender) dan penerima (receiver). Dalam MPI, prinsip ini diwujudkan dalam komunikasi point-to-point:

   send() → mengirim pesan ke proses tertentu.

   recv() → menerima pesan dari proses tertentu.

   Struktur Umum:

```
comm.send(data, dest=rank_tujuan)
data = comm.recv(source=rank_sumber)
```

   Komunikasi Point-to-point mirip dengan “surat elektronik” Dimana Pengirim menentukan tujuan dan penerima membaca pesan dari sumber tertentu.

b. Komunikasi Kolektif

   Selain komunikasi antar dua proses, MPI juga mendukung komunikasi kelompok, disebut collective communication Dalam *parallel database* (Taniar, 2008), operasi *reduce()* sering digunakan untuk menghitung total agregat dari hasil query yang dibagi di beberapa node

| Fungsi | Deskripsi |
|---|---|
| bcast() | Mengirim data dari satu proses ke semua proses lain. |
| scatter() | Membagi data menjadi beberapa bagian ke banyak proses. |
| gather() | Mengumpulkan data dari semua proses ke satu proses pusat. |

| | |
|---|---|
| reduce() | Melakukan operasi agregasi (penjumlahan, minimum, maksimum, dll.) pada data dari semua proses. |

c. Model Komunikasi MPI

MPI memiliki dua model komunikasi utama, yang diturunkan dari konsep sistem jaringan komputer (Davies, 2019):

| Jenis | Deskripsi | Kelebihan | Kekurangan |
|---|---|---|---|
| Synchronous | Pengirim *menunggu* sampai penerima siap menerima pesan. | Aman dan determinis tik. | Kurang efisien karena proses bisa menunggu lama. |
| Asynchronous | Pengirim tidak menunggu; pesan dikirim ke buffer dan eksekusi lanjut. | Efisien, paralel penuh. | Perlu pengaturan sinkronisasi agar pesan tidak hilang. |

3. Mpi4py adalah *wrapper* resmi untuk Python yang mengimplementasikan seluruh fungsi standar MPI. Mpi4py mendukung komunikasi antar proses dengan Pythonic syntax, serialisasi otomatis menggunakan pickle, dukungan penuh untuk *NumPy arrays* untuk operasi numerik cepat. Contoh Struktur Program MPI di Python:

```
from mpi4py import MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if rank == 0:
    data = {"msg": "Halo dunia paralel!"}
    comm.send(data, dest=1)
elif rank == 1:
    data = comm.recv(source=0)
    print(f"Proses {rank} menerima data: {data}")
```

4. Komunikasi Antar Node: Send dan Receive

Menurut Gordon Davies (2019) dalam *Networking Fundamentals*, pertukaran data antar node mengikuti prinsip dasar komunikasi jaringan: *source*, *destination*, *protocol*, dan *message integrity*. Dalam konteks MPI, hal ini direpresentasikan dengan:

- send(): proses pengirim menyalurkan pesan ke proses tujuan.
- recv(): proses penerima menunggu dan mengambil pesan dari pengirim.

C. Langkah Praktikum

1. Eksperimen 1: Hello World Paralel

Tujuan: mengenal konsep rank dan komunikasi dasar antar proses.

```
file: hello_mpi.py
from mpi4py import MPI
```

```
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print(f"Halo dari proses {rank} dari total {size} proses.")
```

Jalankan:

```
mpiexec -n 4 python hello_mpi.py
```

2. Eksperimen 2: Komunikasi Antar Node (Send & Receive)
   Tujuan: memahami cara kerja komunikasi point-to-point antar proses.

```
# file: send_receive.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = "Pesan dari proses 0"
    comm.send(data, dest=1)  # kirim ke proses 1
    print(f"[{rank}] Mengirim: {data}")
elif rank == 1:
    data = comm.recv(source=0)  # terima dari proses 0
    print(f"[{rank}] Menerima: {data}")
```

Jalankan:

```
mpiexec -n 2 python send_receive.py
```

3. Eksperimen 3: Pengiriman Data ke Banyak Proses
   Tujuan: melakukan komunikasi satu ke banyak (*broadcast*).

```
# file: broadcast_example.py
from mpi4py import MPI

comm = MPI.COMM_WORLD
rank = comm.Get_rank()

if rank == 0:
    data = "Data dari proses 0"
else:
    data = None

data = comm.bcast(data, root=0)
print(f"[{rank}] menerima data: {data}")
```