BAB III
METODE PENELITIAN

3.1 Jenis dan Pendekatan Penelitian

Penelitian ini merupakan penelitian eksperimental dengan pendekatan kuantitatif. Menurut Sugiyono (2020), penelitian kuantitatif dilakukan untuk mengetahui pengaruh suatu perlakuan terhadap variabel lain dalam kondisi yang terkontrol. Pendekatan kuantitatif digunakan karena berorientasi pada data numerik yang dapat diolah secara statistik untuk menjawab rumusan masalah penelitian secara objektif. Dalam konteks ini, seluruh pengujian dilakukan melalui pengukuran terukur seperti waktu eksekusi, penggunaan CPU, dan konsumsi memori pada masing-masing bahasa pemrograman.

Selaras dengan itu, Sadish et al. (2002) menekankan bahwa penelitian eksperimental harus dirancang dengan prinsip *controlled variation* yakni setiap perubahan hasil harus dapat ditelusuri ke perlakuan yang diberikan. Dalam penelitian ini, perlakuan yang dimaksud berupa variasi bahasa pemrograman (Go, Rust, Java, dan Python) serta jumlah thread atau *goroutines* yang dijalankan.

Pendekatan eksperimental dipilih karena memungkinkan pengamatan langsung terhadap pengaruh perubahan konfigurasi sistem terhadap performa eksekusi program secara terukur. Sementara pendekatan kuantitatif dipilih karena hasil penelitian berupa data numerik (waktu, memori, dan CPU usage) yang dapat dianalisis menggunakan metrik *speedup ratio* dan *efficiency* untuk menilai efektivitas masing-masing bahasa dalam mengimplementasikan *concurrent* dan *parallel programming*.

3.2 Kerangka Kerja Penelitian

Kerangka kerja penelitian ini menggambarkan tahapan-tahapan sistematis yang dilakukan peneliti untuk memperoleh data, menguji hipotesis, dan menarik kesimpulan secara ilmiah. Penelitian ini dirancang dengan pendekatan eksperimental, sehingga setiap tahap berfokus pada kontrol variabel, replikasi pengujian, serta analisis kuantitatif terhadap hasil pengukuran performa sistem. Secara umum, penelitian ini terdiri atas lima tahapan utama, yaitu:

3.2.1 Studi Literatur

Tahapan awal yang bertujuan untuk mengidentifikasi teori dasar, model pemrograman, serta hasil penelitian terdahulu yang relevan dengan topik *concurrency* dan *parallelism*. Literatur yang digunakan mencakup teori dari Amdahl, (1967); Gustafson, (1988); Roscoe, (2005), serta penelitian empiris seperti Abhinav et al., (2020); Akoushideh & Shahbahrami, (2022); Costanza et al., (2019); Michailidis & Margaritis, (2012). Hasil dari tahap ini digunakan untuk merumuskan variabel eksperimen dan memilih algoritma *benchmark* yang representatif.

3.2.2 Perancangan Eksperimen

Pada tahap ini ditentukan:

1. Variabel bebas: bahasa pemrograman (Go, Rust, Java, dan Python) serta jumlah thread/goroutines (1, 2, 4, 8, dan 16).
2. Variabel terikat: waktu eksekusi, pemanfaatan CPU, penggunaan memori, dan rasio *speedup*.

3. Variabel kontrol: algoritma dan dataset yang sama untuk tiap bahasa, lingkungan uji yang identik, serta versi compiler dan interpreter yang dikendalikan.

Desain penelitian menggunakan model eksperimen faktorial 4×5, di mana empat bahasa pemrograman diuji dengan lima variasi jumlah thread. Analisis hasil didasarkan pada Hukum Amdahl dan Gustafson untuk menilai efisiensi dan skalabilitas sistem.

3.2.3 Implementasi Benchmark

Setiap algoritma diimplementasikan dengan logika yang identik dalam empat bahasa pemrograman berbeda:

1. Go: menggunakan *goroutines* dan *channels* (model CSP).
2. Rust: menggunakan *async/await* dan pustaka *Tokio* untuk *task scheduling*.
3. Java: menggunakan *Fork/Join Framework* dan *ExecutorService*.
4. Python: menggunakan *multiprocessing* dan pustaka *NumPy*.

Tiga algoritma uji yang digunakan meliputi:

1. *Matrix Multiplication*: *compute-bound workload*.
2. *Parallel Sorting*: *mixed workload*.
3. *Producer–Consumer Pattern*: *I/O-bound workload*.

Ketiganya dipilih karena mewakili pola kerja paralel yang umum digunakan dalam sistem multi-core modern.

3.2.4 Pelaksanaan Pengujian

Eksperimen dilakukan pada lingkungan sistem yang dikontrol, menggunakan perangkat keras dengan spesifikasi konstan dan sistem operasi tunggal. Setiap kombinasi konfigurasi dijalankan sebanyak lima kali pengulangan, dan nilai median digunakan untuk mengurangi pengaruh *noise* sistem. Data dikumpulkan menggunakan alat ukur seperti:

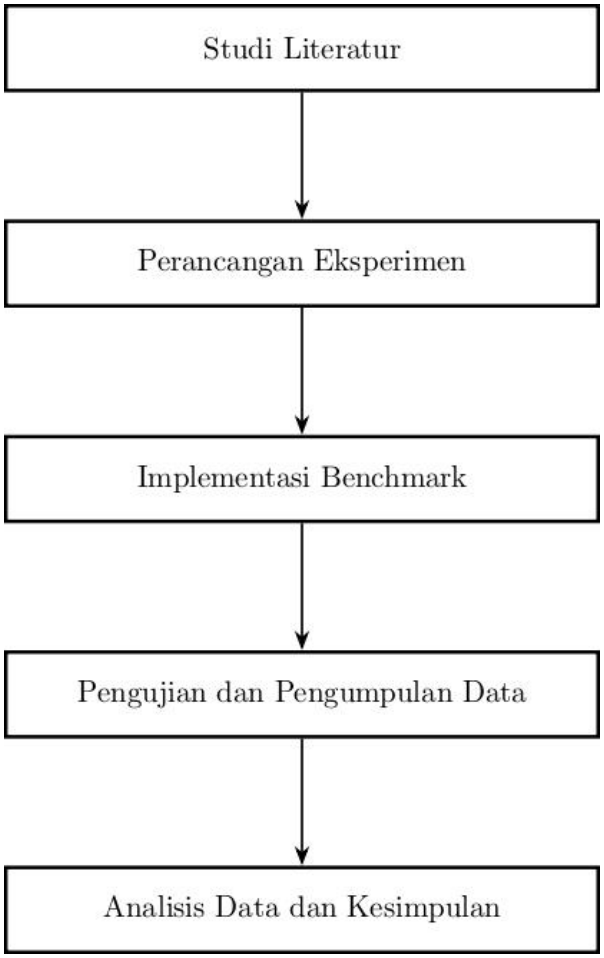
a. /usr/bin/time -v untuk waktu eksekusi dan penggunaan memori.
b. perf stat -d untuk siklus CPU, *cache misses*, dan efisiensi pemrosesan.

Mode *CPU governor* diset ke performance agar frekuensi prosesor tetap stabil selama pengujian.

3.2.5 Analisis dan Interpretasi Hasil

Data hasil pengujian dianalisis menggunakan pendekatan kuantitatif deskriptif dan komparatif. Langkah-langkah analisis mencakup:

1. Perhitungan *mean*, *median*, dan *standard deviation*.
2. Perhitungan *speedup (S(p))* dan *efficiency (E(p))* terhadap baseline (1 thread).
3. Pembuatan grafik hubungan *jumlah thread vs waktu eksekusi* dan *speedup vs efficiency*.
4. Interpretasi hasil dengan membandingkan teori concurrency dan parallelism di Bab II.



Gambar 1. Diagram Alur Kerja Penelitian

3.3 Uraian Kerangka Kerja Penelitian

3.3.1 Studi Literatur

Tahap ini bertujuan untuk memperoleh landasan teoritis dan pemahaman menyeluruh mengenai konsep *concurrency* dan *parallelism*, model pemrograman yang digunakan, serta hasil penelitian terdahulu yang relevan. Kegiatan yang dilakukan pada tahap ini meliputi:

1. Mengumpulkan dan mempelajari sumber pustaka seperti buku, artikel ilmiah, serta prosiding konferensi yang berkaitan dengan pemrograman konkuren dan paralel.

2. Mengidentifikasi teori dasar seperti Amdahl's Law (1967) dan Gustafson's Law (1988) sebagai acuan untuk mengukur batas kecepatan sistem paralel.
3. Menganalisis penelitian terdahulu seperti Abhinav et al. (2020), Costanza et al. (2019), Michailidis & Margaritis (2012), dan Akoushideh & Shahbahrami (2022) yang membahas implementasi *concurrency* pada berbagai bahasa pemrograman.

Hasil dari tahap ini digunakan untuk menentukan arah penelitian, memilih algoritma uji (*benchmark*), dan merumuskan variabel eksperimen.

3.3.2 Perancangan Eksperimen

Tahap ini berfokus pada perumusan desain penelitian dan variabel yang akan diuji. Langkah-langkah yang dilakukan meliputi:

3.3.2.1 Penetapan Algoritma Benchmark

Algoritma *benchmark* yang digunakan meliputi:

1. Matrix Multiplication, representasi *compute-bound task*.
2. Parallel Sorting, representasi *mixed workload*.
3. Producer–Consumer Pattern, representasi *I/O-bound task*.

Ketiga algoritma ini dipilih karena mewakili pola kerja umum dalam sistem komputasi paralel dan konkuren.

3.3.2.2 Penentuan Variabel Penelitian

Penelitian ini menggunakan tiga jenis variabel, yaitu:

1. Variabel bebas: bahasa pemrograman (Go, Rust, Java, Python) dan jumlah thread/goroutines (1, 2, 4, 8, 16).

2. Variabel terikat: waktu eksekusi, pemanfaatan CPU, penggunaan memori, dan rasio speedup.
3. Variabel kontrol: algoritma, ukuran input data, lingkungan sistem, serta versi compiler/interpreter yang digunakan.

3.3.2.3 Rancangan Desain Eksperimen Faktorial 4×5

Bahasa	Threads	Repetisi	Ukuran Input
Go	1-16	5 ×	512-4096
Rust	1-16	5 ×	512-4096
Java	1-16	5 ×	512-4096
Python	1-16	5 ×	512-4096

Tabel 1. Rancangan Desain Eksperimen

Desain eksperimental ini mengombinasikan empat bahasa pemrograman dan lima variasi jumlah thread/goroutines. Setiap kombinasi dijalankan sebanyak lima kali pengulangan untuk memperoleh hasil median yang stabil dan mengurangi *noise* sistem.

3.3.2.4 Penentuan Metode Analisis Data

Analisis hasil dilakukan menggunakan metrik *speedup* dan *efficiency* berdasarkan Hukum Amdahl dan Gustafson untuk mengukur *scalability* serta *parallel efficiency*.

$$S(p) = \frac{T_1}{T_P} \; ; \; E(p) = \frac{S(p)}{p}$$

dengan:

T_1 = waktu eksekusi pada 1 thread (*baseline*),

T_p = waktu eksekusi pada p thread,

$S(p)$ = *speedup* pada jumlah thread p,

$E(p)$ = efisiensi paralel sistem.

3.3.3 Implementasi Benchmark

Tahap ini merupakan proses penerjemahan rancangan eksperimen ke dalam implementasi kode nyata. Setiap algoritma diimplementasikan menggunakan gaya dan pustaka *concurrency/parallelism* yang idiomatik untuk masing-masing bahasa, yaitu:

- Go: menggunakan *goroutines* dan *channels* (CSP model).
- Rust: menggunakan *async/await* dan pustaka *Tokio* untuk *task scheduling*.
- Java: menggunakan *Fork/Join Framework* serta *ExecutorService*.
- Python: menggunakan *multiprocessing* dan pustaka *NumPy*.

Setiap implementasi dioptimalkan untuk menjaga kesetaraan logika dan algoritmik, sehingga hasil perbandingan antar bahasa dapat dianggap adil (*fair comparison*).

3.3.4 Pengujian dan Pengumpulan Data

Tahap pengujian dilakukan dengan menjalankan seluruh implementasi pada perangkat keras dan sistem operasi yang sama untuk menjamin konsistensi hasil. Langkah-langkah pengujian meliputi:

1. Menjalankan setiap konfigurasi eksperimen sebanyak lima kali pengulangan.

2. Menggunakan alat ukur performa sistem, antara lain:
    a. `time -v` untuk waktu eksekusi dan penggunaan memori.
    b. `perf stat -d` untuk pengukuran siklus CPU, *cache misses*, dan efisiensi eksekusi.
3. Menyimpan hasil dalam format log dan mengekstrak nilai median dari tiap pengukuran.
4. Menetapkan *CPU governor* pada mode **performance** untuk menjaga kestabilan frekuensi prosesor selama eksperimen berlangsung.

Hasil dari tahap ini berupa dataset performa numerik yang siap untuk dianalisis.

3.3.5 Analisis Data dan Kesimpulan

Tahap ini bertujuan untuk menganalisis dan menginterpretasikan hasil pengujian menggunakan pendekatan kuantitatif. Langkah-langkah analisis meliputi:

1. Menghitung nilai rata-rata, median, dan *standard deviation* untuk setiap konfigurasi.
2. Menghitung *speedup ratio (S(p))* dan *efficiency (E(p))* berdasarkan hasil baseline (1 thread).
3. Membuat visualisasi berupa grafik hubungan antara jumlah thread dan waktu eksekusi, serta grafik *speedup–efficiency*.
4. Membandingkan hasil dengan teori Amdahl dan Gustafson guna menilai skalabilitas performa.

5. Menarik kesimpulan mengenai efektivitas implementasi *concurrency* dan *parallelism* pada masing-masing bahasa pemrograman.

Tahap ini menghasilkan interpretasi akhir yang menjadi dasar penyusunan kesimpulan penelitian pada Bab V.

3.4 Tempat Penelitian

Penelitian ini dilaksanakan di lingkungan pengembangan lokal menggunakan perangkat keras dan perangkat lunak yang dikonfigurasi secara terkontrol untuk menjamin konsistensi hasil eksperimen. Seluruh pengujian dilakukan secara *offline* untuk menghindari interferensi proses eksternal dan memastikan hasil yang stabil serta dapat direplikasi. Sistem yang digunakan berbasis Ubuntu 24.04 LTS (Noble Numbat) sebagai platform modern dengan dukungan penuh terhadap komputasi multi-core dan *multi-threaded execution*.

3.4.1 Spesifikasi Perangkat Keras

Komponen	Spesifikasi
Perangkat	ASUS Vivobook S 14 (Model M5406UA)
Prosesor	AMD Ryzen 7 8845HS (8 cores / 16 threads, hingga 5.1 GHz)
GPU	AMD Radeon 780M (Integrated RDNA3)
Memori (RAM)	16 GB LPDDR5X @ 7467 MHz
Penyimpanan	512 GB NVMe SSD
Resolusi Layar	1920 × 1200 @ 60 Hz
Mode Daya	*Performance mode* (CPU governor: performance)

Tabel 2. Perangak Keras Praktikan

3.4.2 Spesifikasi Perangkat Lunak

Komponen	Versi / Keterangan
Sistem Operasi	Ubuntu 24.04.3 LTS (Linux kernel 6.14.0-33-generic)
Desktop Environment	GNOME 46.2 (Wayland)
Bahasa Pemrograman	Go 1.22, Rust 1.75, Java 21 (OpenJDK), Python 3.12
Compiler / Toolchain	GCC 14, binutils 2.42, glibc 2.39 *(toolchain default Ubuntu 24.04)*
IDE / Editor	Neovim 0.9.5 Build type: Release LuaJIT 2.1.1703358377 dan terminal Z shell 5.9
Alat Analisis	/usr/bin/time, perf stat, htop, sysbench

Tabel 3. Perangkat Lunak Praktikan

Berdasarkan catatan rilis Ubuntu 24.04 LTS, versi bawaan toolchain adalah rustc 1.75, Go 1.22, GCC 14, dan Python 3.12 (Caconical, 2024). Versi-versi ini dipilih untuk menjamin kestabilan dan kompatibilitas dengan ekosistem *multi-core development* modern.

3.4.3 Lokasi dan Kondisi Eksperimen

Seluruh eksperimen dilakukan di sistem pribadi peneliti dengan kondisi lingkungan terkontrol sebagai berikut:

1. Semua proses dijalankan dalam mode performance untuk menjaga frekuensi prosesor konstan selama uji.
2. Aplikasi latar belakang non-esensial (browser, editor, media player) dinonaktifkan selama pengujian.

3. Setiap konfigurasi dijalankan sebanyak lima kali, dan hasil median digunakan untuk mengurangi pengaruh *noise*.
4. Semua hasil eksekusi disimpan dalam bentuk *log file* dan diolah menggunakan skrip Python untuk analisis kuantitatif.

Penelitian ini dilaksanakan pada bulan Oktober 2025, menggunakan seluruh perangkat keras dan perangkat lunak dalam kondisi terkini (stable release), agar hasil pengujian mencerminkan performa lingkungan pengembangan modern berbasis arsitektur *multi-core*.

3.5 Teknik Analisis Data

Analisis data pada penelitian ini dilakukan menggunakan pendekatan kuantitatif deskriptif dan komparatif, dengan tujuan menilai performa dan efisiensi implementasi *concurrency* serta *parallelism* pada empat bahasa pemrograman modern: Go, Rust, Java, dan Python.

Data yang dianalisis meliputi waktu eksekusi, pemanfaatan CPU, dan penggunaan memori dari setiap kombinasi konfigurasi eksperimen. Setiap konfigurasi dijalankan sebanyak lima kali pengulangan, dan nilai median digunakan sebagai representasi hasil akhir untuk mengurangi pengaruh *noise* sistem atau fluktuasi beban CPU.

3.5.1 Analisis Kuantitatif Deskriptif

Tahap ini bertujuan untuk menggambarkan karakteristik dasar dari hasil eksperimen. Data performa dikumpulkan dalam format numerik dan diolah menggunakan *Python script* dengan pustaka NumPy dan Pandas. Langkah-langkah analisis deskriptif mencakup:

1. Menghitung nilai rata-rata (*mean*), median, serta standar deviasi (*standard deviation*) untuk tiap kombinasi variabel.
2. Menampilkan hasil dalam bentuk tabel performa per bahasa pemrograman dan per jumlah thread.
3. Membuat visualisasi menggunakan Matplotlib untuk menunjukkan hubungan antara jumlah thread dengan waktu eksekusi, *speedup*, dan efisiensi.

3.5.2 Analisis Perbandingan (Comparative Analysis)

Analisis perbandingan dilakukan untuk menilai perbedaan performa antar bahasa pemrograman terhadap beban dan kondisi eksekusi yang identik. Langkah ini dilakukan dengan menghitung rasio performa relatif antara setiap bahasa menggunakan persamaan berikut:

$$R_{i,j} = \frac{T_i}{T_j}$$

dengan:

T_i= waktu eksekusi bahasa pemrograman ke-i,

T_j= waktu eksekusi bahasa pemrograman ke-j,

$R_{i,j}$= rasio perbandingan performa antara dua bahasa pemrograman.

Nilai $R_{i,j} > 1$ menunjukkan bahwa bahasa j lebih cepat daripada i, sedangkan $R_{i,j} < 1$ menandakan sebaliknya. Analisis ini membantu mengidentifikasi bahasa dengan efisiensi eksekusi terbaik pada tiap jenis workload (*compute-bound*, *I/O-bound*, atau *mixed*).

3.5.3 Analisis Skalabilitas

Analisis skalabilitas dilakukan untuk menilai sejauh mana kinerja meningkat seiring bertambahnya jumlah thread atau goroutines yang dijalankan. Dua metrik utama digunakan dalam tahap ini: speedup dan efficiency, berdasarkan Amdahl's Law dan Gustafson's Law.

$$S(p) = \frac{T_1}{T_p}; E(p) = \frac{S(p)}{p}$$

dengan:

T_1= waktu eksekusi pada satu thread (*baseline*),

T_p= waktu eksekusi pada $p$thread,

$S(p)$= *speedup* terhadap baseline,

$E(p)$= efisiensi sistem paralel terhadap jumlah thread.

Interpretasi:

$S(p) \approx p \rightarrow$ *linear scalability*,

$E(p) > 0.8 \rightarrow$ efisiensi sangat baik,

$E(p) < 0.5 \rightarrow$ terdapat overhead tinggi atau *bottleneck concurrency*.

3.5.4 Visualisasi dan Interpretasi Hasil

Hasil analisis disajikan dalam bentuk grafik dan tabel agar pola performa lebih mudah diamati. Visualisasi dilakukan menggunakan pustaka Matplotlib dan Seaborn dengan tiga bentuk utama:

1. Grafik hubungan antara jumlah thread (X) dan waktu eksekusi (Y),
2. Grafik hubungan antara jumlah thread (X) dan speedup (Y),
3. Grafik hubungan antara jumlah thread (X) dan efficiency (Y).

Interpretasi dilakukan dengan cara:

1. Menganalisis tren peningkatan atau penurunan kinerja tiap bahasa terhadap pertambahan thread.
2. Mengamati titik jenuh (*saturation point*) di mana penambahan thread tidak lagi meningkatkan performa.
3. Mengaitkan hasil empiris dengan teori concurrency dan parallelism pada Bab II, serta hukum Amdahl dan Gustafson sebagai kerangka konseptual.

3.5.5 Kesimpulan Akhir Analisis

Tahap akhir ini bertujuan untuk menarik simpulan kuantitatif dari seluruh hasil analisis, meliputi:

1. Bahasa pemrograman dengan performa terbaik pada tiap jenis workload.
2. Bahasa dengan efisiensi tertinggi dalam pemanfaatan *multi-core processor*.
3. Pola skalabilitas masing-masing bahasa dan keterbatasannya.
4. Kesesuaian hasil empiris dengan teori skalabilitas paralel.
5. Kesimpulan dari tahap ini menjadi dasar utama penyusunan hasil dan pembahasan pada Bab IV, serta rekomendasi pengembangan sistem berbasis *concurrent* dan *parallel programming* di masa mendatang.