Chapter 3: Parallelization

Parallelization Pattern Overview

In the previous chapters, we've explored Prompt Chaining for sequential workflows and Routing for dynamic decision-making and transitions between different paths. While these patterns are essential, many complex agentic tasks involve multiple sub-tasks that can be executed *simultaneously* rather than one after another. This is where the **Parallelization** pattern becomes crucial.

Parallelization involves executing multiple components, such as LLM calls, tool usages, or even entire sub-agents, concurrently (see Fig.1). Instead of waiting for one step to complete before starting the next, parallel execution allows independent tasks to run at the same time, significantly reducing the overall execution time for tasks that can be broken down into independent parts.

Consider an agent designed to research a topic and summarize its findings. A sequential approach might:

1. Search for Source A.
2. Summarize Source A.
3. Search for Source B.
4. Summarize Source B.
5. Synthesize a final answer from summaries A and B.

A parallel approach could instead:

1. Search for Source A *and* Search for Source B simultaneously.
2. Once both searches are complete, Summarize Source A *and* Summarize Source B simultaneously.
3. Synthesize a final answer from summaries A and B (this step is typically sequential, waiting for the parallel steps to finish).

The core idea is to identify parts of the workflow that do not depend on the output of other parts and execute them in parallel. This is particularly effective when dealing with external services (like APIs or databases) that have latency, as you can issue multiple requests concurrently.

Implementing parallelization often requires frameworks that support asynchronous execution or multi-threading/multi-processing. Modern agentic frameworks are

designed with asynchronous operations in mind, allowing you to easily define steps that can run in parallel.

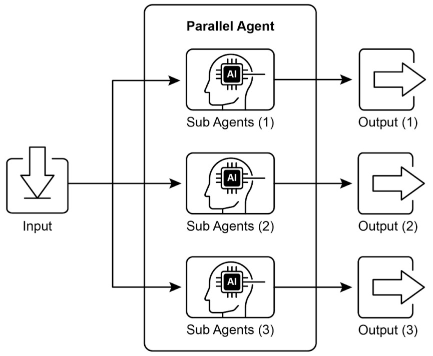

Fig.1. Example of parallelization with sub-agents

Frameworks like LangChain, LangGraph, and Google ADK provide mechanisms for parallel execution. In LangChain Expression Language (LCEL), you can achieve parallel execution by combining runnable objects using operators like | (for sequential) and by structuring your chains or graphs to have branches that execute concurrently. LangGraph, with its graph structure, allows you to define multiple nodes that can be executed from a single state transition, effectively enabling parallel branches in the workflow. Google ADK provides robust, native mechanisms to facilitate and manage the parallel execution of agents, significantly enhancing the efficiency and scalability of complex, multi-agent systems. This inherent capability within the ADK framework allows developers to design and implement solutions where multiple agents can operate concurrently, rather than sequentially.