Parallelims and Pipelining

by David G. Messerschmitt

Supplementary section for Understanding Networked Applications: A First Course, Morgan Kaufmann, 1999.

Concurrency is a key to achieving high throughput for repetitive tasks. Assigning separate tasks to different hosts, they can execute concurrently and possibly increase task throughput. There are two common ways to do this: parallelism and pipelining.

Parallelism

Parallelism is illustrated in Figure 1., where tasks are assigned to dedicated active objects, which may in turn interact with passive objects. If the tasks are completely independent—the active objects do not need to interact, and they do not share common passive objects—each active object and its associated passive objects can execute in a different host. The tasks are then said to be executing in parallel. The task throughput can be increased in this way, since the total throughput is the sum of the throughputs of the individual hosts.

Example: If a WWW server is overloaded, two or more parallel WWW server's running in parallel can be set up, each handling a fraction of the users. As long as the servers are presenting static (unchanging) information, this will work fine. If the information is dynamic (changing) or if the actions of one client are intended to affect others, complications arise.

Pipelining

Pipelining is used for repetitive tasks where each task can be broken down into independent sub-

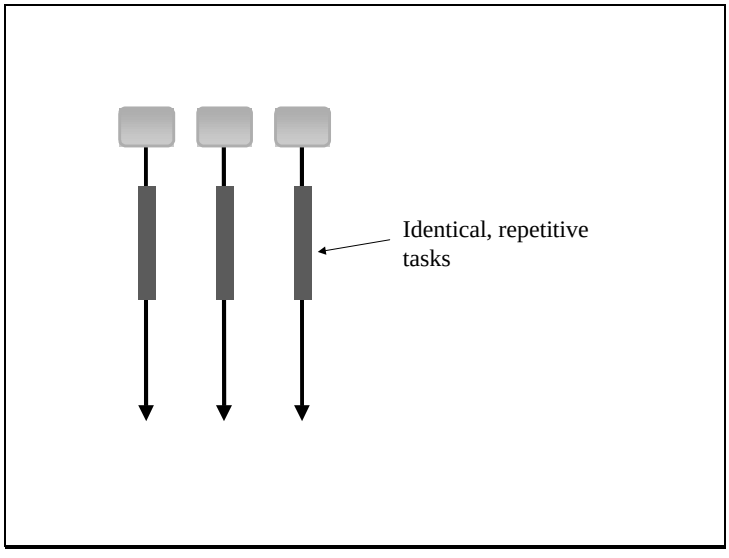

Figure 1. Independent tasks can be executed in parallel on active objects.

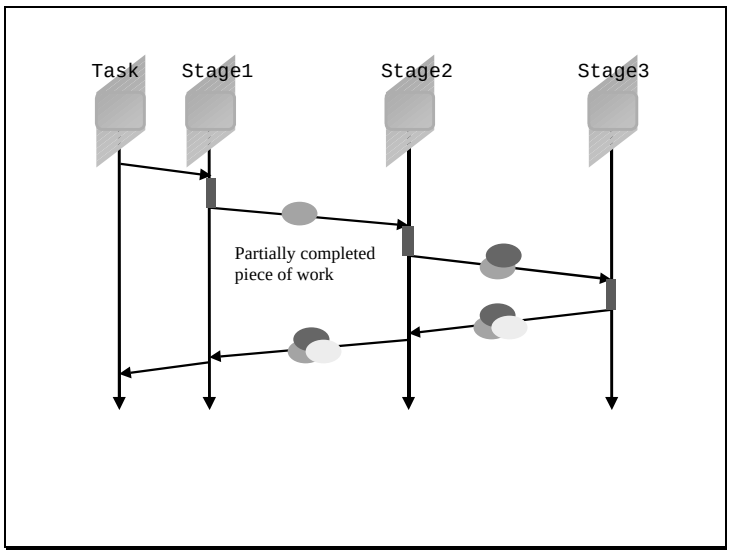

Figure 2. Completion of a task by stage using message with reply interaction.

tasks (called *stages*) which must be performed sequentially, one after the other. The parallelism approach to such repetitive staged tasks would be to assign an active object to each task. The pipelining alternative is to assign an object to each *stage* of the task, so that multiple objects collaborate on each task.

A straightforward approach is to again assign each task to an active object, which passes the task to a passive object handling the first stage, which in turn passes the task to a second-stage object, etc., as shown in Figure 2.. This approach is just a variation on parallelism, where each task is partitioned across staged passive objects. The pipelining variation is shown in Figure 3., where each task is delegated (using a message rather than method invocation) to the next stage after completion of the previous stage. As illustrated, concurrency is achieved by assigning the repetitive tasks to the same active and passive objects, passing a new task each time the previous task completes. There is concurrency because at each point in time the different stages are completing on their respective stages for *different* tasks.

Analogy: In the physical world, the manufacturing of a product assembled from components has this staged structure. For example, if the entire task is assembly of one automobile, its stages are assembling the body, adding wheels, adding the engine, etc.

The assembly line, a major innovation in the industrial age, using pipelining. The simple idea is to specialize workers to each stage of assembly (analogous to specializing objects to each stage of a task), position those workers along the assembly line in the order of their stages, and then move the partially assembled product by them. At one instance of time, multiple cars are being assembled, a different one at each stage of assembly.

As this analogy suggests, the advantage of pipelining over parallelism is *specialization*. The task can be partitioned and assigned to objects according to the specialized stages, and those stages can be kept busy by performing only their stage on repetitive tasks. Similar specialization in the parallelism case shown in Figure 2. results in the specialized stage objects being blocked. This specialization is not an issue in the software portion of an application (although it is important to computer hardware architects), but becomes significant when the application integrally incorpo-

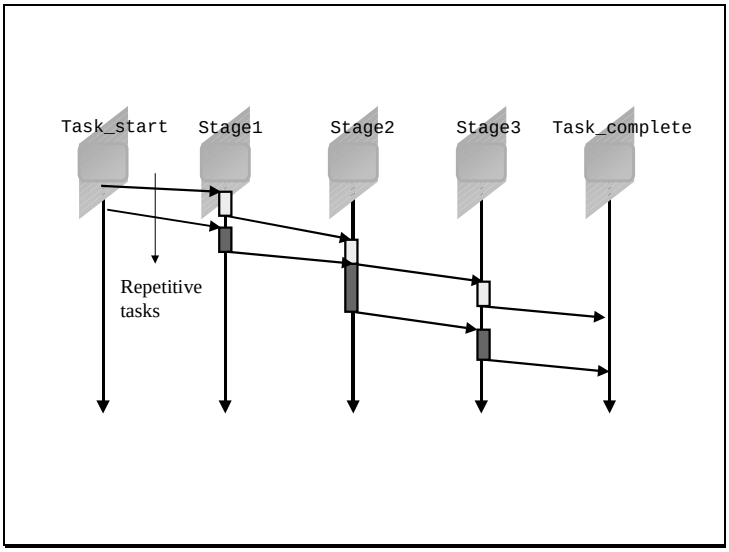

Figure 3. Same situation as Figure 2., except using delegation by message.

rates users.

Example: In a workflow application, such as the purchasing department, each worker can be assigned complete tasks and work in parallel. However, there are benefits to specializing workers to particular stages of the task (like identifying a vendor), in which case pipelining is the appropriate form of concurrency. It keeps the workers busy.

Discussion

D1 Staying within the realm of lawmaking, what is the role of concurrency in the governmental legislative process. What are examples for parallelism and pipelining? What are analogies to active and passive objects?

Review

Two forms of concurrency are parallelism and pipelining. Parallelism assigns different tasks to different concurrent objects. Pipelining divides a repetitive task into specialized stages, and assigns those stages to different objects. Concurrency in pipelining results from allowing the different stages of successive tasks to be overlapped in time. Pipelining is of primary interest in workflow applications (and incidentally hardware design) because it allows the specialization of concurrent workers.

Exercises

E1. Discuss which objects should be active and which should be passive—or either would be acceptable—for the following situations:

a. Two parallel tasks

b. Two pipelined multiple-staged tasks

c. Two concurrent tasks assigned to different hosts

d. Two concurrent tasks assigned to the same host

E2. Give two examples of pipelining in the physical world not mentioned in the chapter.

E3. For each of the following techniques, describe under what conditions task processing throughput can be increased by that technique, and how it would be done:

a. Concurrency

b. Pipelining

c. Parallelism

d. Adjusting the task completion time in the presence of congestion (irregular task arrivals and/or processing time)

e. Multitasking

f. Multithreading