2.1 Lecture Summary

2 Functional Parallelism

2.1 Future Tasks

Lecture Summary: In this lecture, we learned how to extend the concept of asynchronous tasks to *future tasks* and *future* objects (also known as *promise* objects). Future tasks are tasks with return values, and a future object is a "handle" for accessing a task's return value. There are two key operations that can be performed on a future object, *A*:

1. *Assignment* — *A* can be assigned a reference to a future object returned by a task of the form, *future* { ⟨ *task-with-return-value* ⟩ } (using pseudocode notation). The content of the future object is constrained to be *single assignment* (similar to a final variable in Java), and cannot be modified after the future task has returned.
2. *Blocking read* — the operation, *A.get()*, waits until the task associated with future object *A* has completed, and then propagates the task's return value as the value returned by *A.get()*. Any statement, S, executed after A.get() can be assured that the task associated with future object A must have completed before S starts execution.

These operations are carefully defined to avoid the possibility of a race condition on a task's return value, which is why futures are well suited for functional parallelism. In fact, one of the earliest use of futures for parallel computing was in an extension to Lisp known as MultiLisp.

Optional Reading:

1. Wikipedia article on Futures and promises.

Mark as completed