- Concurrency platform -> provide a layer of software that coordinates, schedules, and manages the parallel-computing resources.
- Dynamic multithreading model -> a simple extension of our serial programming model; allows the programmer to specify only the logic parallelism within a computation and the threads within the underlying concurrency platform schedule, which load-balance the computation themselves. That is, it allows programmers to specify parallelisms in applications without worrying about communication protocols, load balancing, and other annoying stuff in static-thread programming.
- P773 The concurrency platform contains a scheduler, which load-balances the computation automatically, thereby greatly simplifying the programmer's chore.
- P773 two features supported by dynamic-multithreading environments: 1. Nested parallelism 2. Parallel loops
- Nested parallelism allows a subroutine to be "spawned", allowing the caller to proceed while the spawned subroutine is computing its result.
- A parallel loop is like an ordinary for loop, except that the iterations of the loop can execute concurrently.
- P774 four advantages of dynamic multithreading model
- Four concurrency keywords: parallel, spawn, sync, new
- P776 The semantics of a spawn differs from an ordinary procedure call in that the procedure instance that executes the spawn - the **parent** - may continue to execute in parallel with the spawned subroutine - its **child** - instead of waiting for the child to complete, as would normally happen in a serial execution.
- P777 circled paragraph
- A procedure cannot safely use the values returned by its spawned children until after it executes a **sync** statement.
- In addition to explicit synchronization provided by the **sync** statement, every procedure executes a **sync** implicitly before it returns, thus ensuring that all its children terminate before it does.
- P777 A model for multithreaded execution: it helps to think of a **multithreaded computation** - the set of runtime instructions executed by a processor on behalf of a multithreaded program - as a directed acyclic graph G = (V,E), called a **computation dag**.
- Instructions involving parallel control are not included in strands, but are represented in the structure of the dag.
- In general case, the set V forms the set of **strand**s, and the set E of directed edges represents **dependencies** between strands induced by parallel control where (u,v) ∈ E means that strand u must execute before strand v.
- P778 If G has a directed path from strand u to strand v, we say that the two strands are **(logically) in series**. Otherwise, strands u and v are **(logically) in parallel**.
- We can picture a multithreaded computation as a dag of strands embedded in a tree of procedure instances.
- Spawn edges and call edges point downward, continuation edges point horizontally to the right, and return edges point upward. Strand u spawning strand v differs from u calling v in that a spawn induces a horizontal continuation edge from u to the strand u' following u in its

procedure, indicating that u' is free to execute at the same time as v, whereas a call induces no such edge.

- Take a look at Fig 27.2 P778
- P778 We shall study the execution of multithreaded algorithms on an **ideal parallel computer**, which consists of a set of processors and a **sequentially consistent** shared memory. For dynamic multithreaded computations, which are scheduled onto processors automatically by the concurrency platform, the shared memory behaves as if the multithreaded computation's instructions were interleaved to produce a linear order that preserves the partial order of the computation dag.
- We can measure the theoretical efficiency of a multithreaded algorithm by using two metrics: **work** and **span**.
- The work of a multithreaded computation is the total time to execute the entire computation on one processor. In other words, the work is the sum of the times taken by each of the strands. For a computation dag in which each strand takes constant time, the work is just the number of vertices in the dag.
- The span is the longest time to execute the strands along any path in the dag. Again, for a dag in which each strand takes unit time, the span is the number of vertices on a longest or **critical path** in the dag.
- P780 The work and span provide lower bounds on the running time Tp of a multithreaded computation on P processors: $T_p \geq T_1/P$, $T_p \geq T_\infty$.
- The **speedup** of a computation on P processors is the ratio T_1/T_p. The speed up can be at most P.
- T_1/T_∞ gives the **parallelism** of the multithreaded computation. As a ratio, the parallelism denotes the average amount of work that can be performed in parallel for each step along the critical path.
- As an upper bound, the parallelism gives the maximum possible speedup that can be achieved on any number of processors. The parallelism provides a limit on the possibility of attaining perfect linear speedup. Specifically, once the number of processors exceeds the parallelism, the computation cannot possibly achieve perfect linear speedup.
- P780 circled paragraph
- P781 definition of **parallel slackness** and its representation, read circled paragraph.
- P781 A multithreaded scheduler must schedule the computation with no advance knowledge of when strands will be spawned or when they will complete - it must operate on-line.
- For scheduling, we analyze **greedy schedulers**, it assigns as many strands to processors as possible in each time step.
- P782 complete step & incomplete step
- P782 Theorem 27.1
- P783 Corollary 27.2, 27.3
- P784 Fig 27.3
- P785 "new" keyword
- P787 We must also account for the overhead of recursive spawning when analyzing the span of a parallel-loop construct.
- Running time due to recursive spawning: θ(lg n)
- P787 circled paragraph

- Race condition
- P788 A **determinacy race** occurs when two logically parallel instructions access the same memory location and at least one of the instructions performs a write.
- P789 circled paragraph
- P789 Between a **spawn** and the corresponding **sync**, the code of the spawned child should be independent of the code of the parent, including code executed by additional spawned or called children.
- P791 circled paragraph