Multithreaded Algorithms (Cormen Ch. 27)

- serial algorithms: algorithms that assume single sequential instruction execution
- parallel algorithms: algorithms that can handle concurrent instruction execution (either within a multiprocessor system or across many independent systems)
- shared memory: the ability of each processor to directly access any memory location within a computer capable of parallel processing
- distributed memory: the use of messaging between processors (or systems) to gain access to memory for other processors (or systems)
- static threading: software abstraction of virtual processors (or threads)
- threads: mechanism for multiple concurrent or interleaved process execution
- concurrency platforms: layer of software used to coordinate, schedule and manage parallel-computing resources

Dynamic multithreaded programming

- dynamic multithreading: method of programming which allows for parallelism without the programmer explicitly managing it, i.e. the concurrency platform manages load balancing, communication protocols, and thread management, etc
- subroutines can be spawned and run concurrently
- parallel for loops: for loop where iterations are executed concurrently
- high-level method for parallel programming via dynamic multithreading:
    - programmer only specifies parallel logic
    - uses four additional keywords within pseudocode:
        - parallel: specifies that a loop (or instructions) may be run concurrently (i.e. in a new thread/process)
        - spawn: specifies that a procedure (or instructions) may be run concurrently (i.e. in a new thread/process)
        - sync: specifies that a procedure must wait as necessary for all spawned children to complete before proceeding to the subsequent instruction
        - new: specifies that a new version of data/variable should be created each time execution point is encountered
    - removing keywords results in serial code
    - serial divide-and-conquer lends itself to adaptation for this approach
    - many modern platforms support this
        - Cilk
        - Cilk++
        - OpenMP

- Task Parallel Library
- Threading Building Blocks

Basics

```
parallel_fibonacci(n)
    if n <= 1
        return n
    else x = spawn parallel_fibonacci(n – 1)
        y = parallel_fibonacci(n – 2)
        sync
        return x + y
```

- serialization (of multithreaded algorithm): serial algorithm resulting from the removal of parallel directives
- nested parallelism: result of spawn keyword before a procedure call
- logical parallelism: the use of high-level parallel description language to specify which components of a computation may proceed in parallel
- the scheduler (and potentially parts of a compiler/interpreter) manage the actual runtime parellelism

A model for multithreaded execution

- multithreaded computation: - the set of runtime instructions executed by a processor on behalf of a multithreaded program program
- computation dag - the directed acyclic graph resulting from multithreaded computation
- conceptually vertices are instructions and edges between in instructions indicate that the source instruction must be executed prior to the sink/destination instruction
- strand: sets of instructions without parallel control specifications (parent must have joined due to sync and if a node in a strand has two children, one must have been spawned)
- continuation edge: an edge that connects a strand to its successor strand within the same procedure instance (two edges on the same level connected by execution order)
- spawn edge: an edge from a strand to a new strand after a spawn directive
- call edge: edges representing normal procedure calls which point downward
- return edge: the edge immediately preceding a sync directive when that leads to the reconnection of a strand to the tree of its calling procedure
- initial strand: the starting strand within a computation dag of a parallel computation
- final strand: the end strand within a computation dag modeling a parallel computation
- ideal parallel computer: a computer consisting of a set of processors and sequentially consistent shared memory
- sequentially consistent: parallel processing which may execute out-of-order instructions but always produces the same results as sequential execution

Performance measures

- work: total time to execute computation on one processor
- span: longest time to execute strands along any path within the computation dag (for each strand taking unit time, equivalent to the length of the critical path within the dag)
- critical path: path containing the most number of vertices within the dag
- running time (Tp) of multithreaded computation on P processors:
  - work law: Tp >= T sub 1 / P -> in one step, ideal parallel computer can do at most P units of work, thus in Tp time can do at most Tp*P work. work law follows
  - span law: P-processor ideal parallel computer cannot run any faster than a machine with an unlimited number of processors or unlimited comp can emulate P comp by using P processors -> Tp >= Tinf
- speedup: ratio of T sub 1 / Tp
- linear speedup: when speedup is linear in the number of processors added
- perfect linear speedup: speedup when T sub 1/Tp == P
- parallelism of computation: T sub 1 / Tinf of work to span
- parallel slackness: (T sub 1 / Tinf) / P = T sub 1/(P*Tinf) i.e. the factor by which the parallelism of the computation exceeds the number of processors in the machine

Scheduling

- required for good performance within parallel computers
- scheduler maps strands to processors directly
- on-line operation: the need for the scheduler to schedule computation within strands with no advanced knowledge of when strands will be spawned or when they will complete
- must operate in a distributed fashion, causing thread to load-balance computation
- analysis of online distributed schedulers is difficult
- better to analyze on-line centralized scheduler
- centralized scheduler: scheduler that knows global state of computation at any time
- greedy schedulers: assigns as many strands as possible to processors in each time step
- complete step: within a greedy scheduler, if at least P strands are ready to execute during a given time step and the scheduler assigns P of the ready strands to processors
- complete step: within a greedy scheduler, if fewer than P strands are ready to execute during a given time step and the scheduler assigns each of the P strands to its own processor
- theorem 27.1:
  - on an ideal parallel computer with P processors, a greedy scheduler executes a multithreaded computation with work T sub 1 and span Tinf in time Tp <= T sub 1/ P + Tinf

- corollary 27.2:
  - the running time Tp of any multithreaded computation scheduled by a greedy scheduler on an ideal parallel computer with P processors is within a factor of 2 of optimal
- corollary 27.3:
  - let Tp be the running time of a multithreaded computation produced by a greedy scheduler on an ideal parallel computer with P processors and let Ts and Tinf be the work and span of the computation, repsectively. Then if P << T sub 1/ Tinf, we have Tp ~ T sub / P or equivalently a speedup of approximately P (<< - much less, ~ approximately equal)

Analyzing multithreaded algorithms

- requires analyzing the sequential running time, the span, and parallel slackness and comparing results

Parallel loops

- often managed by compiler which may execute parallel for loop as a divide-and-conquer subroutine with nested parallelism (can result in binary tree of execution with leaves equivalent to a single loop)
- coarse leaves: parallel for loop leaves which contain multiple iterations of the for loop

Race conditions

- deterministic: parallel computation which always performs the same operations given the same input
- nondeterministic: parallel computation which may have varying behavior between runs given the same input
- determinacy race: two parallel instructions access the same memory location and at least one of the instructions performs a write
- independent strands: strands that operate in parallel but contain no determinacy races between them