Lecture 18: Concurrency

COMP 524 Programming Language Concepts
Stephen Olivier
April 14, 2009

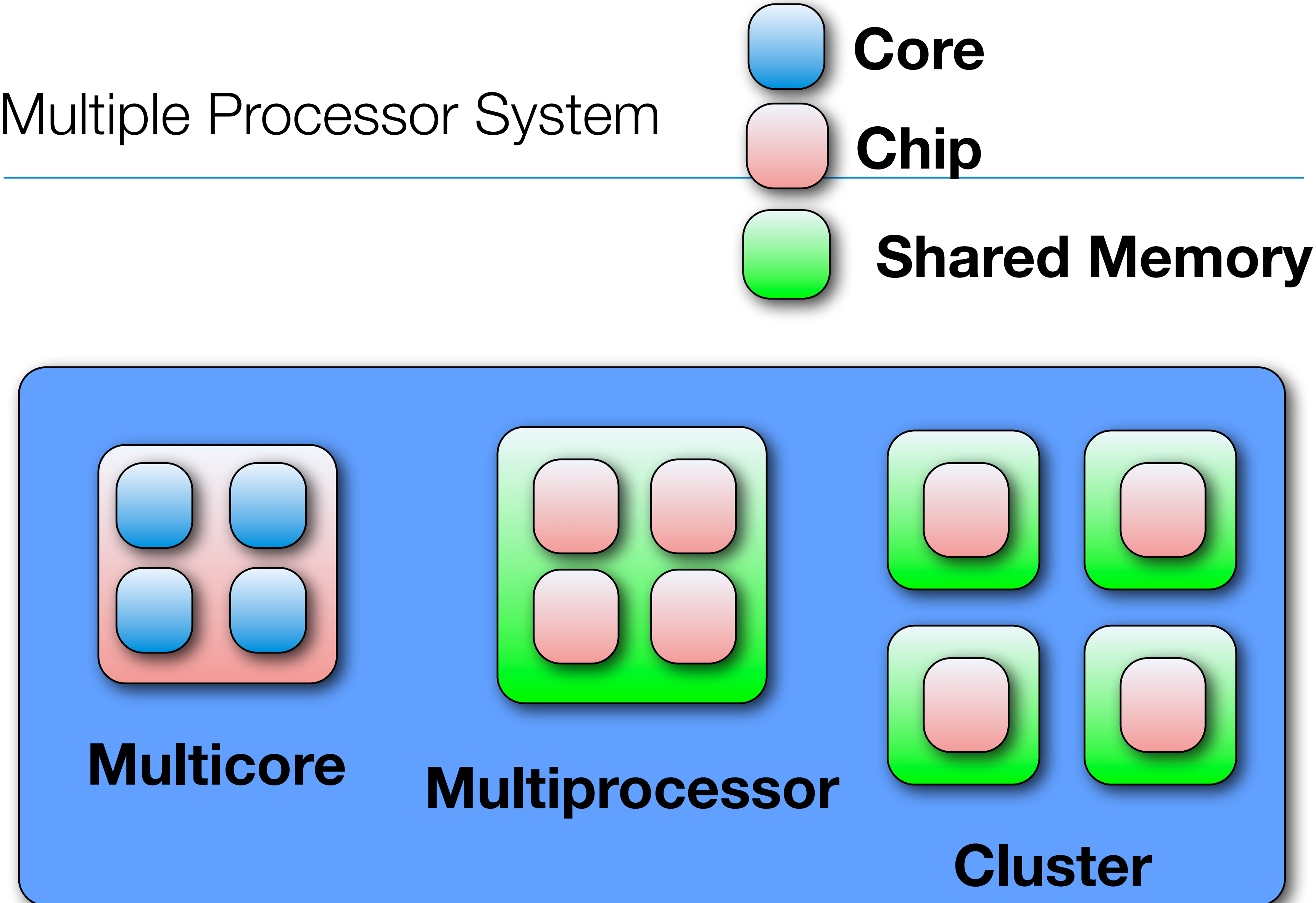
Based on slides by A. Block, notes by N. Fisher, F. Hernandez-Campos, and D. Stotts

The University of North Carolina at Chapel Hill

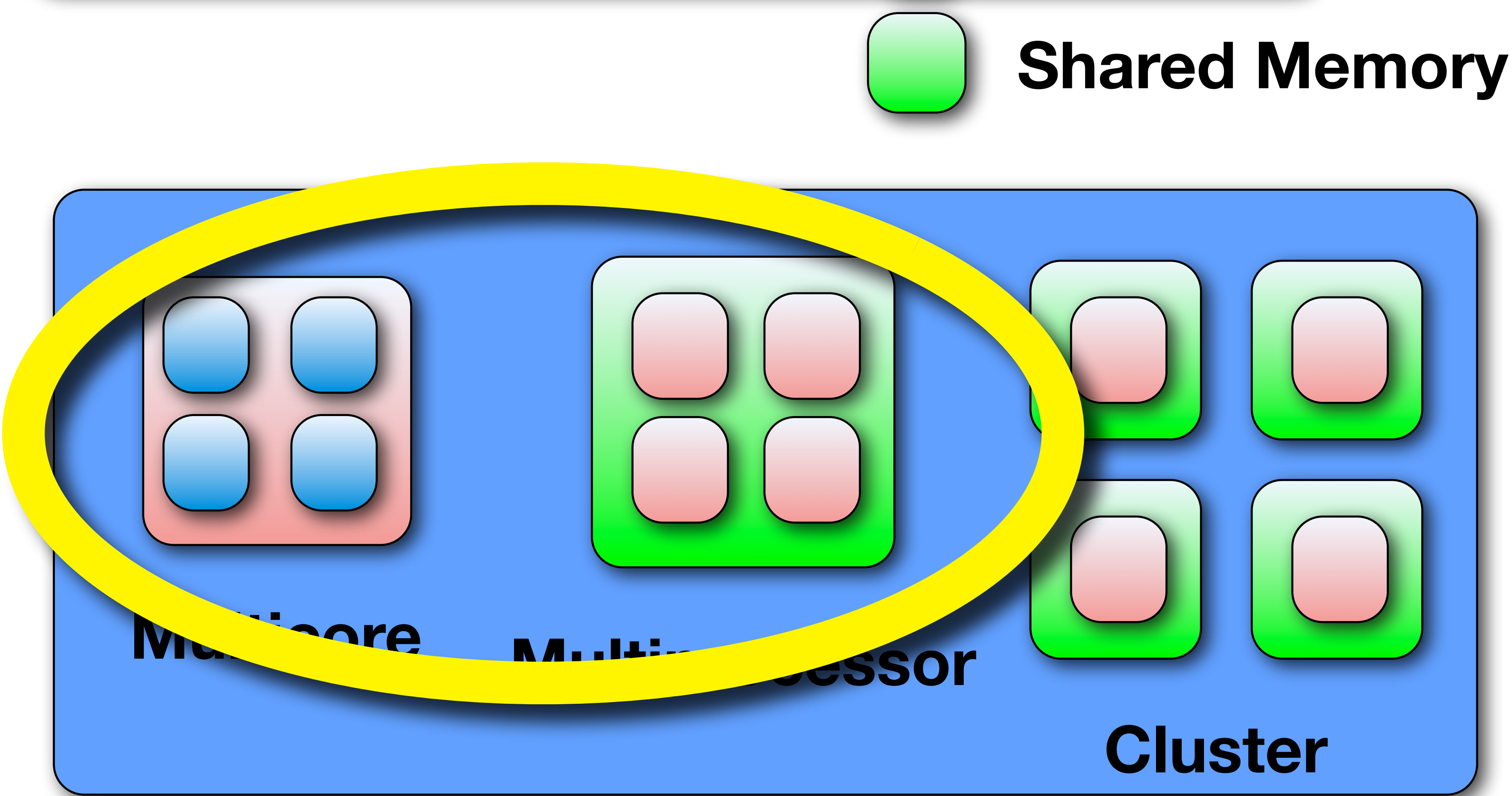
Why Allow for Concurrency?

- Handle multiple events (web server request handling)
- Allow for more effective utilization of physical devices
- Allow for multiple processes to run on multiple processors.

Multiple Processor System



These are both called multiprocessor



Race conditions

- A race condition occurs whenever there is a detrimental way to interleave multiple segments of code.
- Race conditions are one of the hardest issues to do correctly in a concurrent system.
  - Languages and libraries offering guarantees of freedom from race conditions have been the subject of much research
    - Hard to do with good performance

```
read head;
A->next:=head;
head:=A;
```

```
read head;
B->next:=head;
head:=B;
```

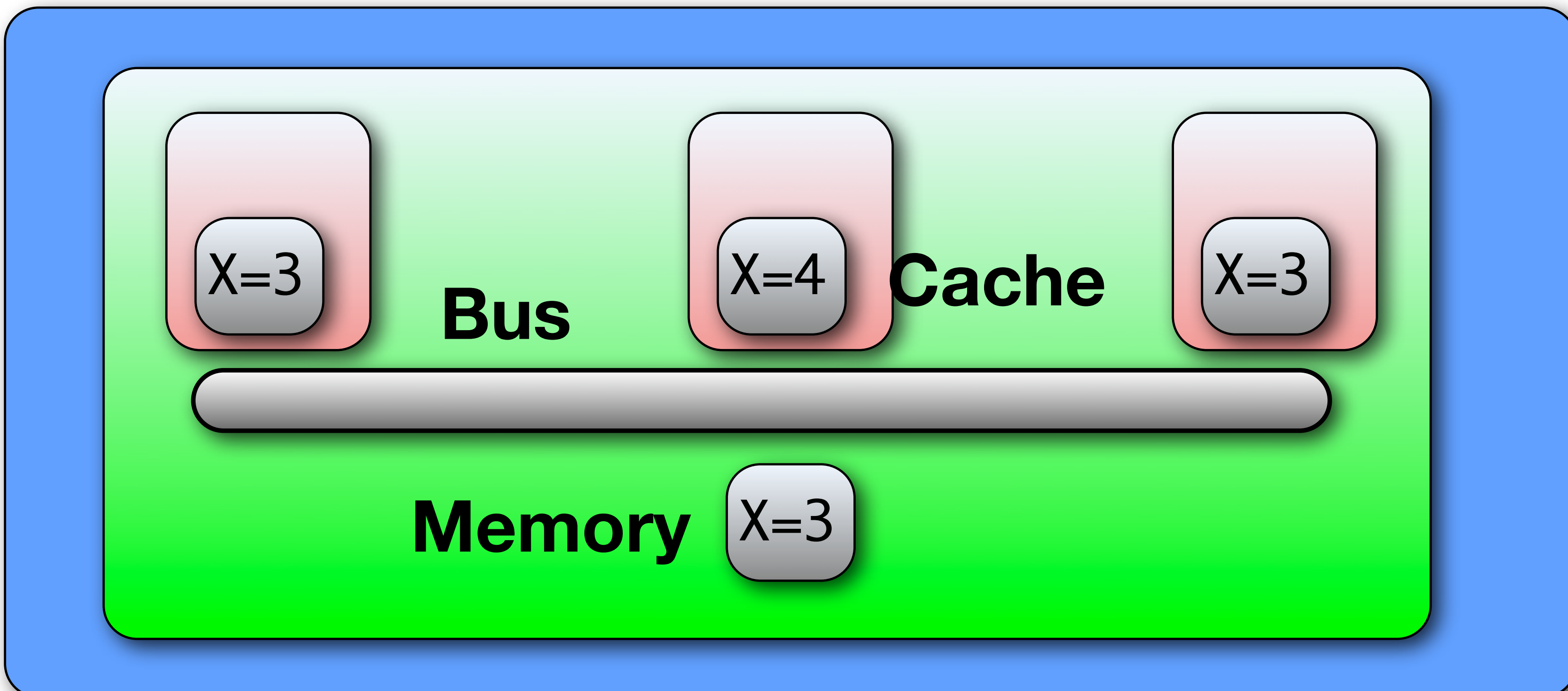
Possible interleaving of the routines above:

```
read head;
A->next:=head;
head:=A;
read head;
B->next:=head;
head:=B;
```

```
read head;
read head;
A->next:=head;
B->next:=head;
head:=A;
head:=B;
```

Cache coherency

- Cache coherency is a problem when multiple processors have their own local copies of data in their cache and values change. (Usually solved in HW)



Parallel Execution

- Heavyweight processes have their own memory space
- Lightweight processes share memory space.
- An execution context in a concurrent system is typically called a **thread**

Creating multiple threads

- Before Java and C#, parallel code consisted of an annotated Fortran or C/C++ with library calls.
  - e.g. OpenMP, MPI
  - Still widely used, especially in scientific & industrial apps
- For Unix, the library for C/C++ is called **POSIX pthreads**.
  - Other frameworks built on top of pthreads
- Microsoft has a similar threading package for Windows

Communication

- Reads and writes into shared memory space
  - Available natively in shared memory systems
  - Supported in some cluster interconnect technologies
- Messages between threads/processes
  - Supported for both shared memory and clusters

Synchronization

- Allows ordering of operations among threads
  - Often needed for program correctness
- May be explicit (by the programmer) or implicit (by the threading library to support higher level abstractions such as loops)
- This is the primary subject of Section 12.3 (read)

Remote Procedure Calls

- Remote Procedure Calls (RPC) are used to communicate between a client and server
- The client calls a local stub, which packages the parameters then sends them to the server and waits for a response.
- Discussed in greater detail in Section 12.4.4 (read)

Six ways to create threads

- co-begin
- parallel loops
- Launch-at elaboration
- fork
- implicit receipt
- Early Reply

co-begin

- Multiple commands can be executed at the same time

```
par begin
  a:=3,
  begin
    c:=4;
    c:c+1
  end,
  b:=4
end
```

parallel loops

- A loop in which iterations execute in parallel

```
co(i:=5 to 10)->
  p(a,b,i)
oc
```

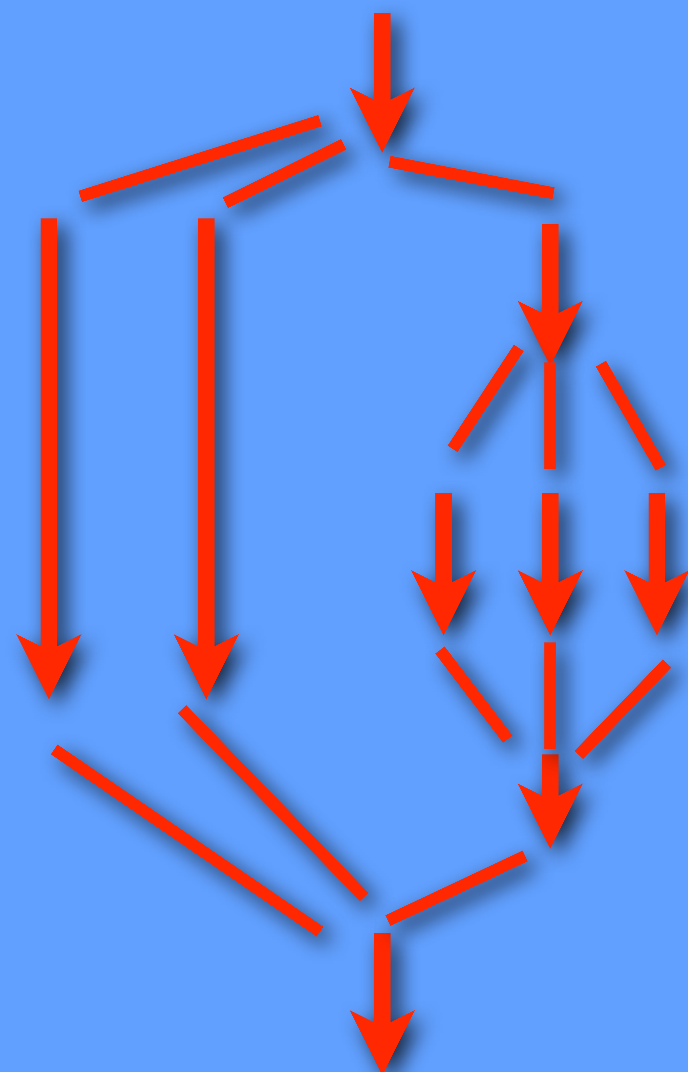
Launch-at-elaboration

- New threads are created when method is launched and destroyed by end of method.

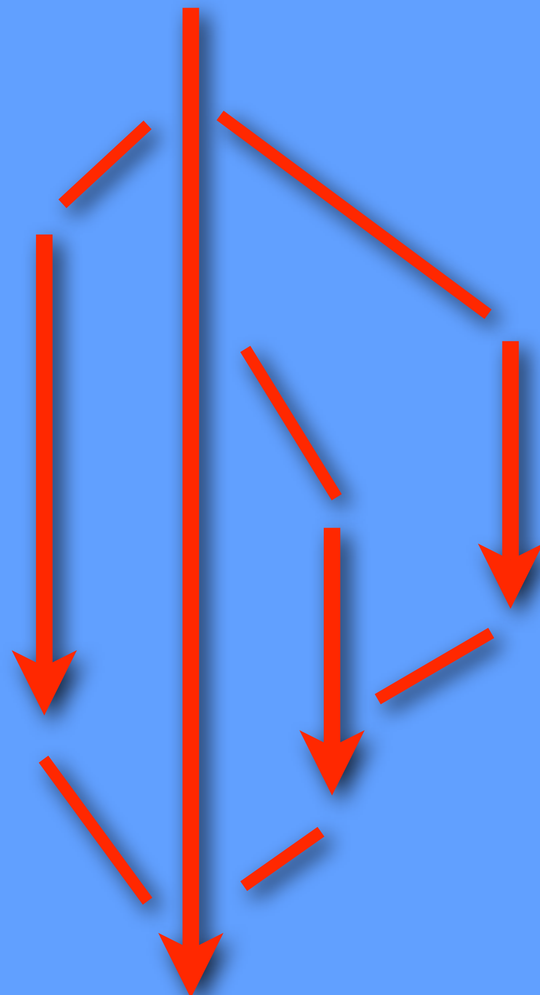
```
procedure P is
  task T is
    ...
  end T;
begin -- P
  ...
end P;
```

Fork/Join

- Threads are created by a function call fork and destroyed by the function call join
  - Allows more general parallelism than some other models
- In Java 5 “forking” is supported by sending tasks to functions that implement the Runnable or Callable interface.



A possible execution using co-begin, parallel loops, or lauch-at-elaboration



A possible execution using explicit fork-join

Fork-join example: Fibonacci in Cilk

- Cilk entends C to support fork-join with spawn & sync

```
cilk int fib(int n)
{
  if (n < 2) return n;
  else {
    int x, y;
    x = spawn fib(n - 1);
    y = spawn fib(n - 2);
    sync;
    return x + y;
  }
}
```

Implicit receipt

- Implicit receipt is similar to a fork except that it causes a new thread to be created in another memory space.
  - Typical model for RPC

Early reply

- Early reply allows for a thread to return a value but continue executing.
    - e.g. Do some work and return result to parent thread, then update some logs

Blocked and runnable

- At any given time a thread is either blocked or runnable.
- A thread is blocked if it is “waiting” for a resource
- Threads that are runnable but not running are enqueued on the ready list.

Preemption

- It is possible for a thread to be “preempted” by another thread
    - e.g., interrupt scheduling

Yielding

- Its possible for a thread to yield
  - Suspends execution and allows another thread to execute
  - Thread state changes from runnable to blocked
- This can cause race conditions
  - Particularly in combination with preemption

Throughput-Oriented Systems

- Want to process events as quickly as possible
  - e.g. Requests to a web server
- Limited communication and synchronization required between threads
  - Concurrent data access issues handled by database system
- Swap threads out while they wait for memory accesses and remote communication
    - Sun Niagara built to support many lightweight threads
    - Cloud computing

Compute-Oriented Systems

- One large program runs on many processors (shared memory and/or a cluster)
  - Typically one thread per processor
- Scientific apps such as climate simulation
- Sometimes require significant communication and synchronization between threads
  - Minimizing communication is typically key to performance

Data Parallel Programming

- SIMD (Single Instruction Multiple Data)
  - Same instructions performed on multiple data simultaneously
  - Developed in the early Cray supercomputers
  - Now built into mainstream processors
    - e.g. 128-bit vector operations in MMX, SSE, Altivec, 3D Now
- SPMD (Single Program Multiple Data)
  - Same program replicated onto multiple threads, each operates on different data (usually based on its thread ID number)
  - e.g. Parallel loops and regions in OpenMP

Task Parallel Programming

- Divide work into a hierarchy of tasks
  - Newly spawned tasks may be moved to idle threads
  - Particularly useful for divide-and-conquer algorithms
- Cilk example given earlier uses this model
- New programming frameworks designed to promote parallel programming for multicore
  - Intel Thread Building Blocks and Ct
  - Microsoft Thread Parallel Library

Programming Language Issues

- Concurrency support in the language or in libraries?
- Do programmers use threads explicitly (e.g. pthreads) or implicitly (e.g. simple forall loop, cilk spawn-sync)?
- Can the compiler provide auto-parallelization (i.e. Intel compiler auto-vectorization for SSE)?
- Can the programmer specify data that is global vs. local or where specific data resides?
- How is synchronization supported?

Performance Issues

- Amdahl's Law
  - Speedup of a parallel program is limited by the time needed for the sequential fraction of the program.
- Load imbalance
  - Uneven distribution of work among processors
- Communication and Memory Operations
  - Latency: time delay to access resource
  - Bandwidth: amount of data transferable per unit time
  - Contention: many threads want to access same resource