Parallel Programming in C++

Weeks 5 and 6

Amittai Aviram – 6 & 13 October 2020 – Boston University

Modern C++ Features

C++ Anonymous Functions

A Convenient Technique

- C++, Python, and other modern languages have syntax for anonymous functions.
- These enable you, e.g., to define a function inline when passing it as an argument to another function.
- Called "lambdas" after *lambda expressions* in the *lambda calculus* (Alonzo Church et al., 1932-36). (Sometimes called "closures," but those also mean something else.)
- Syntax: [capture symbols] (parameters) { body }
- The capture list transfers in-scope symbols into the function's environment.

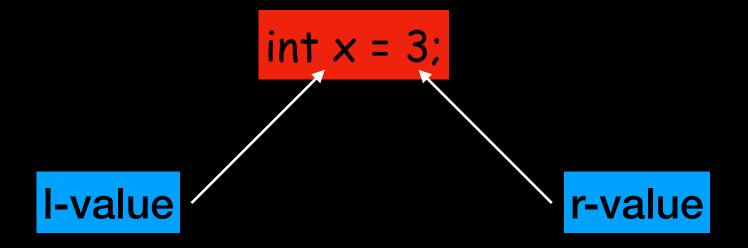
Coding: anonymous functions.

C++ Smart Pointers For Memory Allocation Control

- A smart pointer holds a ("raw") pointer to allocated memory.
- It de-allocates the memory (calling delete) when it goes out of scope.
 - When it goes out of scope, the compiler inserts a call to its destructor.
 - The destructor calls delete.
- Types of smart pointer:
 - unique_ptr allows only one scope to "own" the memory at a time.
 - shared_ptr allows more than one scope to "own" the memory uses a reference counter to call the destructor when the object goes out of the last scope.
 - weak_ptr keeps track of a shared_ptr object.

L-Values and R-Values

C++ Subtleties



You can take the address:

int*
$$p = &x$$
;

You cannot take the address:

You can assign:

$$x = 4$$
;

You cannot assign:

Variables and functions

Literals and temporaries

int x = 3 + y;

L-Value and R-Value References

- A reference works as an alias for an object.
- For "plain old data" (POD) objects, passing by I-value reference and passing by r-value reference are semantically identical.
- For complex objects,
 - Passing by I-value reference does not invoke a constructor.
 - Passing by r-value reference invokes a move constructor, which determines exact behavior.
- std::move casts an object as an r-value reference.

The const and volatile Modifiers

- const makes an object immutable.
- volatile make the compiler treat the object as if assignment to it had a side-effect (such as printing to stdout).

```
template <typename value_t, typename index_t>
value_t sequential_fibonacci(value_t n) {
  // initial conditions
  value_t a_0 = 0;
  value_t a_1 = 1;
  // iteratively compute the sequence
  for (index_t index = 0; index < n; ++index) {
    const value_t tmp = a_0;
    a_0 = a_1;
    a_1 += tmp;
  return a_0;
```

C++ Concurrency

C++ Threads

Basics

- C++ threads are true kernel threads
 - Lightweight
 - In a shared address space
 - Assigned to processors as the latter become available
- Defined in the STL <thread> header
- Launched upon construction

Coding: "Hello, World!"

Returning a Value Two Techniques

- A C++ thread has no mechanism for transferring a function's return value to the shared space using normal function return-value syntax.
- We use function parameters ("out-parameters") on void functions instead:
 - A pointer type holding the *address* in shared memory to which the function writes the return value in shared memory.
 - A *promise*, passed by *r-value reference*, to whose associated *promise* the function writes the return value.

Coding: sequential and parallel Fibonacci numbers.

Static Task Scheduling Schemes to Divide Up the Work

- Block Distribution
 Each thread works on a contiguous block of the shared data.
- Cyclic Distribution
 Each thread *id* works on items items *id*, *id* + *c*, *id* + 2*c*, ...
 This may worsen the problem of *false sharing*.
- Block-Cyclic Distribution
 Each thread has every block of size c in each stride s.
 This generalizes the other two schemes.
 The smaller the block, the better the load balance (roughly).

Dynamic Scheduling

Determined at Runtime

- Each thread grabs a task, finishes it, and grabs the next task.
- Access to the source of tasks must be serialized
 - Lock
 - Condition variable

Synchronization Primitives

Used to Co-ordinate Threads and Access to Shared Data

- Mutual exclusion lock (mutex)
 - Use a closure to release the lock automatically e.g., lock_guard.
- Condition Variable
 - Thread A waits
 - Thread B notifies
- Barrier

Coding: condition variables.

Join

Lock-Free Synchronization

- Atomic types
 - These allow for an atomic change of value, including increment.
- Compare and swap (CAS)
 - Build atop an atomic type.
 - Used in a loop.
 - Say threads A and B want to add a subtotal to a sum. This requires
 - Read
 - Update
 - Write
 - By the time A reads and gets ready to write, B may have written, so the read is stale!
 - A compares its read value with the sum and writes only if these are equal. If not, it tries again.