COMP 322: Parallel and Concurrent Programming

Lecture 38: Concurrent and Parallel Languages and Frameworks

Zoran Budimlić and Mack Joyner
{zoran, mjoyner}@rice.edu

http://comp322.rice.edu

What have we learned in this course?

- Functional programming for parallelism
- Lazy computation, streams
- Futures and promises
- Data-driven programming approach
- Computation graphs and their properties
- Map/Reduce programming model
- Data-parallel programming model
- Loop parallelism
- Locality control
- Handling concurrency while avoiding deadlock/livelock/starvation
- Barrier and point-to-point synchronization
- Actor programming model

Habanero

- Habanero-Java and Habanero-C
- Async/finish, futures/promises, loop parallelism, phasers, locality control, actors, isolation
- HJlib is a library implementation of these features
- Still developed and improved
- Python, Scala, Rust, X10, OpenMP, Chapel, Java, C/C++
- There's also PCDP-Java
  - Coursera equivalent of COMP 322
- No streams

https://habanero.cc.gatech.edu/

X10

- Designed and developed at IBM
- One of the original “Next-generation” Asynchronous Partitioned Global Address Space projects
- Ancestor of Habanero Java
- Originally based on Java, later switched to Scala
- Async, finish, loop parallelism, clocks (phasers), locality control
- No abstract metrics, data-driven execution, actors, streams

http://x10-lang.org/

Chapel

- Designed, implemented and maintained by Cray
- Partitioned Global Address Space
- Loop parallelism, task parallelism
- Locality control
- Distributed system execution
- Tasks, futures, promises
- No phasers, actors, abstract metrics, data-driven execution

https://chapel-lang.org/

Kotlin

- From the creators of IntelliJ
- Based on Java
- Multi-paradigm programming language
  - Functional, object-oriented
- Lots of support for functional programing
- More compact than Java
- Fully interoperable with Java
- Support for coroutines: very similar to asyncs and future tasks
- Low-level synchronization between tasks
- Channels
- No loop parallelism, phasers, abstract metrics, streams, locality control, actors

https://kotlinlang.org/

Go

- Multi-paradigm, object-oriented, concurrent language
- Goroutines (asyncs)
- Channels
- Concurrency control structures
  - Sending messages between coroutines
- No phasers, loop parallelism, futures/promises, abstract metrics, actors, locality control

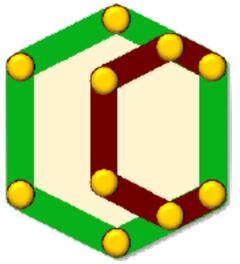
https://go.dev/

Rust

- Multi-paradigm programming language
- Threads
- Message passing
- Shared-state safe concurrency
- Extensible concurrency with Sync (similar to Java Synchronized) and Send traits
- No phasers, loop parallelism, async/finish, futures/promises, actors, locality

https://www.rust-lang.org/

Cilk/Cilk++



- Language developed at MIT
  - Commercialized and bought by Intel
- Task-parallel programming model
- Lots of advances in the work-stealing load balancing runtime techniques
- Implicit “finish” for every function
- No loop parallelism, phasers, abstract metrics, actors

https://cilk.mit.edu/

Python/Ray

- Library based approach
- Aimed at data science, machine learning, data processing
- Futures and actors
- No task-level parallelism on shared memory
- No abstract metrics, phasers, loop parallelism

https://www.ray.io/

Scala

- Functional, Java-based multiparadigm language
- Futures/promises
- Channels
- Data-driven programming model
- Actors
- No abstract metrics, phasers

https://www.scala-lang.org/

Haskell

- Functional programming language
- Lazy computation!
- Haskell threads
- “Pure” parallelism - deterministic
- No race conditions, no deadlocks
- Concurrency between IO and computation
- Synchronizing variables
- Channels, futures, promises
- par, pseq, force functions
- No loop paralelism, abstract metrics, phasers, actors

https://www.haskell.org

CnC

- Data-driven programming model
- Language and library based
- Java, C, C++, Scala, Python
- Tagged computation and data
- Easy to distribute
- Easy to checkpoint/restart
- Locality control
- No phasers, loop parallelism, futures/promises, abstract metrics, actors

https://icnc.github.io/

Intel Threading Building Blocks

- Library-based
- C/C++
- Work-stealing runtime
- Tasks (asyncs), loop parallelism, locality control, concurrency mechanisms
- Parallel reductions, maps, filters
- No futures/promises, abstract metrics, phasers, actors

https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html#gs.xrs06b

Hadoop

- Map/Reduce programming model
- Based on Java
- Distributed programming model for large scale computation
- No shared memory concurrency, async/finish, futures/promises, loop parallelism

https://hadoop.apache.org/

Spark

- Map/Reduce programming model
- Python, SQL, Scala, Java, R
- Distributed programming model for large scale computation
- Highly optimized in-memory computation
- No asyncs, futures/promises, loop parallelism, shared-memory synchronization

https://spark.apache.org/

OpenSHMEM

- Library-based approach
- Partitioned Global Address Space
- Make the distributed memory “look” like a shared memory
- One-sided communication
- Collective (barrier) synchronization
- Locality control, atomic operations
- Mostly C/C++ based
- Implementations for modern supercomputers with modern networking subsystems
- No asyncs, futures/promises, abstract metrics, actors, streams

http://openshmem.org

OpenMP

- Compiler/language extensions for existing languages
- C/C++ and Fortran
- Annotation (pragma) based approach
- Widely supported by modern C++ compilers
- Designed for shared-memory systems
- Loop parallelism, tasks, locality control
- Extensions for GPU programming
- No phasers, abstract metrics, streams, futures/promises, actors

https://www.openmp.org/

MPI

- Library framework
- Message-passing programming model
- Designed for distributed systems
- Implementations on top of several programming languages
  - C/C++
  - Java
  - Fortran
  - Julia, MATLAB, OCaml, Python, R
- Implementations for most modern supercomputers
- No tasking, futures/promises, abstract metrics, streams, phasers
- “MPI + X” is still the most dominant approach, with X being OpenMP most of the time

https://www.open-mpi.org/

Summary

- Concurrent and parallel programming is becoming pervasive
- Many languages and frameworks support some aspects
- Most of them do not support all aspects of concurrent and parallel programming
- It's possible to build additional features on top of a few basic ones
- You have learned most of the basic concepts in COMP 322