CS331: Programming lang Lab

A Sahu

Dept of Computer Science & Engineering

IIT Guwahati

Outline

- Course Structure
- Tutorial and Assignments
- Books and Reference Materials
- Tools to be installed in your PC
- Grading Pattern
- Rules for Malpractices

Course Structure

- Programming paradigms: imperative and declarative (introduction);
- **Concurrent programming**
  - basic idea, Java language introduction,
  - concurrent programming with Java (threads and libraries);
- **Logic programming**
  - basic idea, Prolog introduction,
  - logic programming with Prolog;
- **Functional programming**
  - basic idea, introduction to LISP/Haskell,
  - functional programming with LISP/Haskell;

Course Structure

- Programming paradigms: imperative and declarative (introduction);
- **Concurrent programming**
  - basic idea, Java language introduction,
  - concurrent programming with Java (threads and libraries);
- **Logic programming**
  - basic idea, Prolog introduction,
  - logic programming with Prolog;
- **Functional programming**
  - basic idea, introduction to LISP/Haskell,
  - functional programming with LISP/Haskell;

Tutorial and Assignment

- Tutorial
  - As there is tutorial components in this courses
  - There will be one hour tutorial every week
  - **Attendance is compulsory : carry marks**
- Timing: 8AM to 9AM Wednesday
- Lab hours: reserved for evaluations, no need to be in Lab for whole duration
- Assignments
  - We will float assignments in Web Site and MS Teams
  - You are supposed to code for the assignment and
  - submit before the deadline of the assignments in MS Teams

Tutorial and Assignment

- **Grading Policy**
  - **Assignment 66%, 26% written tests (one premid, one post mid), 8% attendance in Tutorial**
- **Concurrent programming**
  - **CP1:**Basic concurrent programming with Java, [8 Marks]
  - **CP2:**Advanced concurrent programming with Java (threads and libraries); [14 Marks]
- **Logic programming**
  - **LP1:** basic idea, Prolog introduction, [8Marks]
  - **LP2**: logic programming with Prolog; [14 Marks]
- **Functional programming**
  - **FP1:**basic idea, introduction to LISP/Haskell, [8 Marks]
  - **FP2:**functional programming with LISP/Haskell; [14 Marks]

Tools to be installed

- Java Compilers
  - Almost every where we get
- Gprolog: in Ubuntu or Any other OS
  - sudoapt-get install gprolog
- Haskel
  - sudo apt-get install cabal-install

Rules for Malpractices

- Copy cases
  - You require to ensure that : no one copy your code
- Both source and destination guy will get
  - either negative Full Marks for that assignment or F Grade
- MOSS Check: Variable change, comments, code position change

TAs

- Suvarthi Sarkar
- Vasantha Reddy
- Shubhradeep Roy
- Akshaya Bhosale
- Manish Karmakar
- ..two other TAs

Concurrent programming

Concurrent programming

- Programming to Simulate Concurrent behavior of system
  - Multi-threading
  - Doing many task simultaneously
- Platform of Concurrent Programming
  - May be uni-processor
  - May be shared or distributed memory multiprocessor
- Parallel Programming
  - Enhancing performance of application by running program in parallel on Multiprocessor

Process and Thread

- Process
  - A sequential computation with its own thread of control
  - Can be many threads of a Process
- Thread
  - A sequential computation is the sequence of the program points that are reached as control flow through source text
  - Light weight process
  - Many things shared by parent process

Communications

- Exchange of data between threads/processes
  - Either by explicit message passing
  - Or through the values of shared variable
- Between Process
  - Message passing
  - Message Passing Interface : MPI-send(), MPI_recv()
- Between thread
  - through the values of shared variable

Synchronizations

- Relates the thread of one process with others

If p is point in the thread of a process ***P***, and q is point in the thread of another process ***Q***,

Then Synchronization can be used to constrain the order in which ***P*** reached to p and ***Q*** reaches to q.

Synchronization Involves: Exchange of control information between processes.

Time Shared and Multiprogramming

- Time shared programs appears to run in parallel
  - Even if it run on uni-processor system
  - Lets go back to Pentium PC, RR Scheduling
- Interrupts (Hardware)
  - Allowed the activity of a central CPU to be synchronized with data channels.

Implicit Synchronization Example

- No need to specify
- Process networks in Unix (Pipe)

  P1 | P2 | …|Pn

  – Each primitive process does a simple job, perhaps a trivial job

  – but short pipeline of processes can do what would otherwise done by substantial program

- Example

```
$ bc  | number  | speak
$ ls | wc –l
$ ps –A | grep mozilla
```

Concurrency as Interleaving

- Concurrent computation
  - Can be described in terms of events, where an event is an un-interruptible action
  - Event: execution of assignment, call, expr evaluation

Concurrency as Interleaving

- Interleaving: The relative order of atomic events
  - An interleaving of two sequence ***S*** and ***T*** is any sequence ***U*** formed from the events of ***S*** and ***T***
  - Subjected to constraints: events of ***S*** retain their order in ***U*** and so the event of ***T***
- Example: S={a,b,c,d,…}, T={1,2,3,4,..}
  - One U can be {1,a,b,c,d,2,3,4,e,5,f,g..}

Basic Coordination and Synchronization

- Sharing Data
  - Reader and Writer
  - ***1R, 1W, MR***, 1R1W, MR1W, 1RMW, MRMW
  - Synchronization necessary: One process should be writer
  - Mutual Exclusion: Critical Section Problem

More than 1 process and 1 must be writer

- Barrier or Fence
  - Wait until some thing
  - Synchronized
  - Example: Phase wise executions

```
For_all_N_threads DoWork1();
waits(N);
For_all_N_threads DoWork2();
waits(N);
```

Sharing Data: Critical Section

- Sharing Data: Reader and Writer
- Locking and unlocking
  - Mutex
- Hardware Instruction to ensure locking
  - Atomic Instructions: TAS, LL/LD pair, XHNG, SWAP
  - TAS (test and set)
  - TTAS (try, test and set)
  - TTAS with Backup

This part we will discuss towards End of this course (in Nov)

- Atomic Register
- Safe Register
- Relative Power of Sync' operations

Why Concurrency?

- Simulating parallelism
- Performance
  - Most of the Application are parallel,
  - underlying Hardware are parallel (Multicore)

Why Multicore ?

- Multiprocessors are
  - Flexible, programmable, high performance
    - Processor are programmable as compared to ASIC application specific integrated circuit
    - Flexible in terms of portability as compared to ASIC
    - Higher Performance than single processor

Why Multicore ?

- Multiprocessors are likely to be cost/power effective solutions
  - **Share lots of resources**
    - ***Personal room is costlier than dormitory***
    - ***You cannot allocate a Bungalow to each student: it will too costly***
      - ***Hostel room with shared facility is sufficient***
  - Need not require very high frequency to run
  - Lots of replication makes easy to manage and cost effective in design

Multicore Difficulties I

- Multiprocessors are likely to be cost/power effective solutions
  - Because it share lots of resources
    - ***Personal room is costlier than dormitory***
  - Sharing resource arise many other problems
    - Critical Sections
      - Lock and Barrier Design
    - Coherence
      - Shared data at all placed should be same
    - Consistency
      - Order should be similar to serial (ROB)
    - One processor Interference others
      - Share efficiently using some policy

Multicore Difficulties II

- Many applications are highly parallel
  - Take benefit of all parallelism (instruction, data and thread)
  - Most of the coder write sequential code
  - Who will extract parallelism from applications ?
  - There is no successful auto-parallelisation tool till date
    - » Attempts: Cetus, SUIF, SolarisCC
- ***Good news: CNN/DNN python parallel library is quite successful in GPU domain***

Multicore Difficulties III

- Task scheduling in multiprocessors
  - Deterministic task scheduling on multiprocessor with more than 2 processor is NP-Complete problem
  - 4 Tasks (A,B, C and D), 3 Processors
    - {A,B,C,D}{}{}, {A,B,C}{D}{}, ...Exponential Number of Solutions
- Simple Example
  - 8 tasks with execution 2, 4, 8, 5, 6, 4, 3, 20
  - Need to executed non-pre-emptively on two processor P1 and P2
  - So that overall execution time is minimized
  - **Solution : Divide 8 tasks in to two subsets, with difference of their sum is minimized ;Subset Sum Problem**

Thanks