Lecture Notes

CS377 - Parallel Programming
Fall 2006
Marc L. Smith

Thu, Sep. 14, 2006
The Linda Model

• A communication and coordination model for concurrent processes

• Augments any existing sequential programming language

• Consists of

  • Tuple Space -- a global shared memory

  • 4 primitive operations on Tuple Space
The Linda Model

- Tuple Space -- a container of tuples
- tuple -- an ordered sequence of typed values, or value-yielding computations

- a tuple whose values are all computed is passive
- a tuple with at least one value still being computed is active
The Linda Model

• The Linda primitive operations:
  • rd( ) -- “read” -- to match a tuples in TS
  • in( ) -- to match/remove tuples from TS
  • out( ) -- to place new tuples in TS
  • eval( ) -- to create new Linda processes
    (places active tuple in TS)

• first two ops are synchronous (blocking) *
  * non-blocking versions also exist: rdp( ) and inp( )

• last two operations are asynchronous (non-blocking)

• first three operations operate on passive tuples.
The Linda Model

- Tuple Space
  - a distributed shared memory
  - not addressable memory (no pointers to tuples!)
  - an associative memory (tuples are matched)
The Linda Model

• Tuple matching is a **generalization** of how we use hashmaps

• Hashmaps
  • key - value pairs
  • lookup key; return corresponding value

• Tuples
  • multiple keys possible (by position within tuple)
  • multiple corresponding values possible (by position)
C-Linda

- An implementation of the Linda Model
  - Base language: C
  - augmented with rd(), in(), out(), and eval()
  - and predicate operations rdp() and inp()
- Let’s look at some examples!
C-Linda Preliminaries

- To compile and execute C-Linda programs on the RocketCalc cluster, you must do the following:
  - login and issue the following copy commands
  - `$ cp ~mlsmith/.bash_profile .`
  - `$ cp ~mlsmith/.bashrc .`
    (then log out and in again)
  - Upon subsequent logins, you should be able to compile and execute C-Linda programs. You can look in your own ~/sca directory for examples.
C-Linda Preliminaries

- Naming convention: myprog.cl

- Environment variable:
  LINDA_CLC=CDS
  "Code Development System"
  LINDA_CLC=LINDA_TCP
  (true distributed computing across cluster)

- Command to check current environment:
  echo $LINDA_CLC

- LINDA_CLC will determine whether you compile for true
  distributed computing or for using Tuplescope
C-Linda Preliminaries

• to compile: (for CDS or LINDA_TCP)
  clc -o foo foo.cl

• to compile for use with Tuplescope, must be under CDS:
  clc -o foo -linda tuple_scope foo.cl

• To run your program under CDS (with or without CDS):
  ./foo
  (must run from an xterm to use Tuplescope)

• To run your program under LINDA_TCP:
  ntsnet -mp 3 -n 8 foo
C-Linda Preliminaries

• to compile/run the sample hello world program: (CDS)
  
  cd ~/sca/cl-examples
  source ~/bin/set-cds
  clc -o chello.cds chello.cl
  ./chello.cds

  (notice all the processes run on node n1, but the sequence of virtual process numbers is nondeterministic!)

• to compile/run the sample hello world program: (TCP)
  
  source ~/bin/set-linda-tcp
  clc -o chello.tcp chello.cl
  ntsnet -mp 3 -n 8 chello.tcp

  (notice processes run on all four nodes across cluster!)
C-Linda Preliminaries

- to compile/run the sample hello world program using Tuplescope:
  cd ~/.sca/cl-examples
  source ~/.bin/set-cds
  clc -o chello.ts -linda tuple_scope chello.cl
  ./chello.ts

  (notice all the processes run on node n1, but the sequence of
  virtual process numbers is nondeterministic!)

- Tuplescope is a powerful visualization tool to use while developing
  your C-Linda programs!
Producer/Consumer

• Write a C-Linda program that evals two processes: Producer and Consumer

• Each process has it’s own array of n elements.

• Between the two processes, a shared buffer exists that will be used to transfer the contents of the producer’s buffer to the consumer’s buffer, one element at a time
Producer/Consumer

- Here’s the pseudo code for producer and consumer:

//shared variables
int buf, n = 80,  p = 0,  c = 0;

process Producer {
    int a[n];
    while (p < n) {
        << await (p == c); >>
        buf = a[p];
        p = p+1;
    }
}

process Consumer {
    int b[n];
    while (c < n) {
        << await (p > c); >>
        b[c] = buf;
        c = c+1;
    }
}