ECE 451/566 - Introduction to Parallel and Distributed Computing

Linda, JavaSpaces & Jini

Manish Parashar
parashar@ece.rutgers.edu
Department of Electrical & Computer Engineering
Rutgers University
Linda

• What is Linda?
  – Parallel programming language based on C (C-Linda) and Fortran (Fortran-Linda)
  – Combines coordination language of Linda with programming languages of C and Fortran
  – Enables users to create parallel programs that perform on wide range of computing platforms
  – Easy to use
  – Based on logically global, associative object memory called tuple space
  – Tuple space provides interprocess communication and synchronization logically independent of the underlying computer or network
  – Implements parallelism with a small number of simple operations on tuple space to create and coordinate parallel processes
  – Commercially available from Scientific Computing Associates Inc.
The Linda Model

- Virtual Shared Memory
- Different parts of the data can reside on different processors.
- Looks like one single global memory space to component processes.
- Linda's Virtual Shared Memory is known as tuple space
- Can be used to implement many different types of algorithms
- Lends itself well to master/worker distributed data structure algorithms
Master/Worker Model using Virtual Shared Memory

- Task and workers are independent of each other
- Master divides work into discrete tasks and puts into global space
- Workers repeatedly retrieve tasks and put results back into global space
- Workers notified of work completion by having met some condition, receiving a "poison pill" or terminated by some other means
- Master gathers results from global space
- Possible ways that tasks can be distributed:
  - Bag of tasks (unordered)
  - Ordered tasks by using a shared counter in tuple space along with task identifiers.
  - Tasks identifiers used to find related data
Linda Basics: Definitions

• Tuple Space
  • Linda's name for its shared data space. Tuple space contains tuples.

• Tuples
  • The fundamental data structure of tuple space.
  • Tuples are represented by a list of up to 16 fields, separated by commas and enclosed in parentheses.
  
    – Examples:
    
      ('arraydata', dim1, 13, 2)
      (var1, var2)
      ('common block', /datacom/)
      ('array sections', a_array(2:10, 4:8))

• Associative Memory Model
  • A tuple is accessed by specifying its contents.
  • From the programmer's point of view, there is no address associated with a tuple.
Linda Basics: Operations

• **Tuple Generation**
  – **out**
    • Generates a data (passive) tuple
    • Each field is evaluated and put into tuple space
    • Control is then returned to the invoking program
    • Example:
      ```
      out ('array data', dim1, dim2)
      ```
  – **Eval**
    • Generates process (active) tuple
    • Control is immediately returned to invoking program
    • Logically, each field is evaluated concurrently by a separate process, and then placed into tuple space. In current implementation, only fields containing function (or subroutine) references result in new processes being created
    • Example:
      ```
      eval ("test", i, f(i));
      ```
Linda Basics: Operations

• Tuple Extraction
  – in
    • Uses a template to retrieve tuple from tuple space.
    • Once retrieved, it is taken out of tuple space and no longer available for other retrievals.
    • If no matching tuple is found, process will block.
    • Provides for synchronization between processes.
    • Example:
      
      in ("arraydata", ?dim1, ?dim2);

  – rd
    • Uses a template to copy data without removing it from tuple space.
    • Once read it is still available for others.
    • If no matching tuple is found, process will block.
    • Example:
      
      rd("arraydata", ?dim1, ?dim2);
Linda Basics: Templates

- Specifies tuple to retrieve
- Consists of sequence of typed fields
- Two kinds of fields
  - Actuals
    - Variables, constants or expression that resolve to constant
  - Formals
    - Holders for data to retrieve.
    - Preceded by a question mark.
    - Assigned values of corresponding fields in matched tuple.

**Example:**
- in("arraydata", ?dim1, ?dim2, ?dim3);
  in ("arraydata", 4, ?dim2, ?dim3);
- Both examples will match the tuple put into tuple space with the following *out* operation:
  - out("arraydata", 4, 6, 8);
Linda Basics: Template Matching

• In order for a template to match a tuple:
  • Have to have the same number of fields.
  • Actuals must have same type, length and values as those in corresponding tuple fields.
  • Formals in template must match type and length of corresponding fields in tuple.
• If several tuples match the template, impossible to predict which will be selected.
• The order of evaluation of fields within a tuple or template is undefined. Therefore, the following constructs should be avoided:
  • `out("string", i++, i);`
    – Can't predict whether i will be incremented before or after it is evaluated for the third field.
  • `in("string2", ?j, ?a[j]);`
    – Can't predict whether j (in third field) will have value set by ?j (second field) or value before statement was executed.
  • `out('string', x, f(x))`
    – In this Fortran example, if the function f() modifies the value of x, we can't predict whether the second field will have the original or the modified value of x.
Linda Basics: Template Matching

• Examples:
  • out ('testdata', i, 3, 4+6)
    – will be matched by:
      • integer cnt, var, sum
      • character*8 string
    .
    .
    .
    • in ('testdata', ?cnt, ?var, ?sum)
  – or
    • in ('testdata', ?cnt, ?var, 10)
  – or
    • in (?string, ?cnt, ?var, ?sum)
Example: C-Linda - Hello World

```c
real_main(argc, argv)
int argc;
char *argv[];
{
    int nworker, j, hello();
    nworker=atoi (argv[1]);
    for (j=0; j < nworker; j++)
        eval ("worker", hello(j));
    for(j=0; j < nworker; j++)
        in("done");
    printf("Hello_world is finished.
");
}

int hello (i)
int i;
{
    printf("Hello world from number %d\n",i);
    out("done");
    return(0);
}
```
Linda Limitations

- High system overhead
- Designed as a parallel computing model, primarily for LANs
  - lack of security model
  - lack of transactional semantics
- Language specific implementation
- Blocking calls, but no notification mechanism
JavaSpaces

• Object coordination system, available as a Jini service
• Based on: David Gelernter’s Linda language
• Model
  – Distributed but share “space”, through which processes can communicate with one another
  – Communication is done by objects
JavaSpaces (Sun Micro.)

- Lightweight infrastructure for network applications
- Distributed functionality implemented through RMI
- Entries written to/from JavaSpace with “write, read”
- “notify” notifies client of incoming entries w/ timeout
- Pattern Matching done to templates with class type comparisons, no comparison of literals.
- Transaction mechanism with a two phase commit model
- Entries are written with a “lease,” so limited persistence with time-outs
Key Features

- All entry fields are strongly typed for matching
- Object model associates behavior with entries
- Matches can return subtypes of template types
- Entries are “leased”; persistence is subject on renewal in order to reduce garbage after failures
- Multiple JavaSpaces cooperate, and transactions span multiple spaces. Partitions provide minimal protection.
- “Eval” functionality is not supported, in order to reduce complexity and overhead
- Transaction model preserves ACID properties
JavaSpace Mechanisms

- Each JavaSpace server exports an object that implements the JS interface locally on the client, and communicates through an implementation specific interface.
- Objects are stored as implementation specific representations, with the serialized class and fields.
- Templates match entries iff each field in template is either null or match the entry field via MarshalledObject.equals. This occurs when the serialized forms of the objects match exactly.
Operations

• Only 6 operations
  – write()
    put an object into space
  – read(), readIfExists()
    get copy of object from space
  – take(), takeIfExists()
    move object from space
  – notify()
    notify about event
Properties

- The Space is:
  - *shared*
    - Processes can access it at the same time;
    - concurrency control is automatic
  - *persistent*
    - An object remains in the space until it is removed!
  - *associative*
    - We find object by their content using template matching
  - *provides transactions*
    - The execution of the 6 operations are atomic;
    - we can use transactions to create composite atomic operations.
  - *Allows changing executable programs*
    - We execute the public methods of the read out objects.

ECE 566: Parallel & Distributed Computing
Associative search

- We use a template to find an Entry object
- A template “matches” an object if:
  1. The template’s type is identical to that of the entry’s or its superclass AND
  2. Each field of the object is identical to the field of the entry
     - a null value matches everything
     - non null value matches only identical value
A simple example

- One process sends a message to the other which will print it out.

    public class Message implements Entry {
        public String text;
    }

    Message msg = new Message();
    msg.text = "Hello";
    space.write(msg, null.Lease.FOREVER);

    Message tmpl = new Message();
    res = space.read(tmpl, null.Lease.FOREVER);
    System.out.print(res.text);
Typical execution patterns

• The execution of parallel and distributed programs follow few typical patterns
  – master-worker
  – command
  – marketplace
  – specialist
  – collaboration
“Master-worker” minta

- The problem is divided into $n$ identical processes (eg. Computing the Mandelbrot-set, or the value of $\pi$)
- Execution pattern
  1. The Master creates $n$ Entry objects and writes them into the space.
  2. The Worker processes read objects from the space, carry out the computation and put the result back to the space
  3. The Master reads the space for partial results, combines them and finishes.
Execution of the “Master-worker” pattern
The marketplace pattern

- In this pattern salespersons (producers) and buyers (consumers) collaborate in the solution of a problem
- Typical application: online auction
- Pattern:
  - Salesperson asks for bids
  - Buyers put in their bids
  - Salesperson chooses the most favourable bid
  - Notifies chosen buyer about the decision
Execution of the “Marketplace” pattern

1. write
2. read
3. write
4. take
5. write
6. take

salesperson

buyer

Accepted bids

Bid requests

bids

ECE 566: Parallel & Distributed Computing
More complicated issues...

- **Lease**
  - Real programs should use the space for a given time (lease period). At the end of the lease, the object is deleted automatically if it is not renewed.

- **Distributed events**
  - The read() and take() operations are blocking. The notify() method can be used to notify processes about an event (read/take/write of an object) of their interest.

- **Transactions**
  - We can group operations into atomic ones preventing data loss in the presence of errors.
In summary

- JavaSpaces helps in creating reliable, fault-tolerant distributed systems.
- The system is simple, the programmer can concentrate on the problem itself.
- The programs are easy to read and understand, elegant and simple to extend.
JavaSpace Limitations

• Simplicity of or lack of security model
• Transactions required for reliable entry reads
• Java RMI = performance bottleneck?
• High overhead from repetitious object serialization
• Currently only a specification exists, but no implementation
Jini: Introduction

• Jini is a general infrastructure for distributed computing between devices on a network
  "Jini technology provides simple mechanisms which enable devices to plug together to form an impromptu community - a community put together without any planning, installation, or human intervention."

• Released by Sun in January 1999
Jini Scenarios

• Digital camera and pictures
  – Camera finds network printer to print pictures
  – Camera finds network disk drive to save pictures
  – Camera turns on lights in the room before taking a picture

• Palm Pilot
  – Student uses Palm pilot to select classes and downloads class schedule into Palm pilot calendar
Jini Overview

• Services
  – A service is an entity that can be used by a person, program or another service
    – a computation, storage, communication channel to another user, a software filter, a
      hardware device, another user
  – Lookup Service
    – Is used to find services in a djinn (a Jini system)
  – Leasing
    • A lease provides access to a service for a fixed time period
    • If a lease is not renewed at the end of the lease period, then the user and provider of
      the service can free all resources connected the lease
  – Transactions
    • Provides protocol for a two-phase commit process for managing state changes
      between objects in a Jini system

• Events
  • Distributed events
  • Distributed version of listeners or Observer-Observable
Jini Overview

• Discovery
  • Service provider looks for a lookup service to register itself

• Join
  • Service provider is registered in the lookup service
  • Service provider registers:
    – A remote reference to itself
    – Descriptive attributes about the service

• Lookup
  • Client requests a service by Java type and/or by attributes
  • Client receives remote reference to the service
Jini Services

• Service
  – An entity that can be used by a person, program or another service

• Examples of a Service
  • Storage
  • A computation
  • Software filter
  • Hardware device
  • A user

• Standard Jini Services
  • Lookup Service, Transaction Manager, JavaSpaces Service
TSpaces (IBM Almaden)

- A set of network communication buffers that work primarily as a global lightweight database system or data repository
- Operators include blocking and non-blocking versions of read and take, write, set operators scan and count, and synchronization operator Rhonda
- Interfaces with data management layer to provide persistent storage and data indexing and query.
- Dynamically modifiable behavior
Key Features

• Database indexing and arbitrary “match, index, and, or” queries
• Transaction layer for data consistency
• Matching available on simple types
• New operators can be downloaded to TSpace and used immediately
• User and group permissions can be set on a Tuplespace and operator basis
• Event register informs clients of events
• HTTP server interface for debugging and maintenance purposes
• Support for large objects through URL reference
TSpace Applications

• Goal: To provide a common platform for linking all system and application services
• Printing services example:
  – heterogeneous machines search in set of TSpaces for a print server of the desired type.
  – Jobs are then “written” as tuples to the local TSpace.
  – Printer client “takes” tuples off local TSpace to process
• Collaboration services:
  – whiteboard: single or multiple clients “write” changes in tuples to one TSpace while all clients “read” tuples
  – video/audio conferencing: discretize multimedia stream into tuples, and “read” or “taken” via central TSpace
• TCP/SLIP Proxy for thin-clients (PalmPilot)
TSpaces vs. JavaSpaces

- Simple types and objects as tuple fields
- No replication, 1 server per TSpace
- Access Control Lists on users and groups
- Event Register invoked on all events
- Database indexing and range queries
- Downloadable operators

- Only serializable objects allowed
- Servers can be replicated across nodes
- Protective partitioning using multiple JSpaces
- Notify () is only invoked for committed writes