Linda

1999.7.23.
S.I.Jun
ICU

Contents

• Linda Concepts and Model
• Linda Examples
• Implementation of Linda
• Conclusion
• Reference
Linda

Developed at Yale University by D. Gelernter, N. Carriero et al.
Goal: simple language-and machine-independent model for parallel programming
Model can be embedded in host language
  – C/Linda
  – Modula-2/Linda
  – Prolog/Linda
  – Lisp/Linda
  – Fortran/Linda

Linda's Programming Model

• Sequential constructs
  – Same as in base language
  – Process creation and coordination are orthogonal to the base language

• Parallelism
  – Sequential processes

• Communication
  – Tuple Space: shared memory supports distributed data structures and facilitate uncoupled programming (sender and receiver need not know each other)
Linda's Tuple Space

- Tuple-Space
  - box with tuples (records)
  - shared among all processes
  - addresses associatively (by contents)

- Atomic operations on Tuple-Space (TS)
  OUT adds a tuple to TS
  READ reads a tuple (blocking)
  IN reads and deletes a tuple

TS Operation

- Four Basic operation
  1) out(t) adds tuple t to TS; The invoking process continues immediately.
  2) in(s) withdraws from TS a tuple t that matches template s.; The invoking process suspends until one matching t becomes available in TS; one is chosen arbitrarily if multiple matching t’s are in TS.
  3) read(s) is the same as in(s), except that the matched tuple t remains in TS.
  4) eval(t) is the same as out(t), except that t is evaluated after it enters TS; It forks a process to perform the evaluation. When the computation of t completes, it turns into a data tuples.

- Two variants (inp, rdp)
  - inp and rdp are predicate versions of in and read. If not found, return 0; otherwise return 1 and a matched tuple.
Linda's Tuple Space (Cnt'd)

- Each operation provides a mixture of
  - actual parameters (values)
  - formal parameters (typed variables, ?)
- Example
  age: integer;
  married: boolean;

  OUT("jones", 31, true)
  READ("jones", ? &age, ? &married)
  IN("smith", ? &age, false)

Atomicity

- Tuples cannot be modified while they are in TS
- Modifying a tuple:
  IN("smith", ? &age, ? &married) /* delete tuple */
  OUT("smith", age+1, married) /* increment age */

- The assignment is atomic
- Concurrent READ or IN will block while tuple is away
Distributed Data Structures in Linda

- Data structure that can be accessed simultaneously by different processes
- Correct synchronization due to atomic TS operations
- Contrast with "centralized manager" approach, where each processor encapsulates local data

Replicated Workers Parallelism

- Popular programming style, also called task-farming
- Collection of $P$ identical workers, one per CPU
- Worker repeatedly gets work and executes it
- In Linda, work is represented by distributed data structure in TS
Advantages Replicated Workers

- Scales transparently -- can use any number of workers
- Eliminates context switching
- Automatic load balancing

Example: Matrix Multiplication

\( C = A \times B \)

- Source matrices:
  
  ("A", 1, A's first row)
  
  ("A", 2, A's second row)
  
  ...
  
  ("B", 1, B's first column)
  
  ("B", 2, B's second column)

- Job distribution: index of next element of C to compute
  
  ("Next", 1)

- Result matrix:
  
  ("C", 1, 1, C[1,1])
  
  ...
  
  ("C", N, N, C[N,N])
**Code for Workers**

```plaintext
repeat
    in("Next", ? &NextElem);
    if NextElem < N*N then
        out("Next", NextElem + 1) i, j
        i = (NextElem -1)/N + 1 Ne = 1 --- (1,1)
        j = (NextElem -1)%N + 1 Ne=10 --> (1,10)
        read("A", i, ? &row) Ne=15 --> (2,5)
        read("B", j, ? &col)
        out("C", i, j, DotProduct(row, col)) Ne=40 --> (4,10)
end
```

**Code for OUTPUT**

```plaintext
// get results from TS and print
for(i=1;i<N;i++)
    for(j=1;j<=N;j++)
        in("C", i, j, Product[i][j]);
print(Product);
```
Example 2: Traveling Salesman Program

- Use replicated workers parallelism
  - A master process generates work
  - The workers execute the work
  - Work is stored in a FIFO job-queue
- Also need to implement the global bound

TSP in Linda
Global Bound

Use tuple ("min", value) representing global minimum
Initialize:
   out("min", maxint)

Atomically update minimum with newvalue:
   in("min", ? & oldvalue);
   value = minimum(oldvalue, newvalue)
   out("min", value)

Read current minimum:
   read("min", ? &value)

Job Queue in Linda

Add a job:
   in("tail", ? &tail)
   out("tail", tail+1)
   out("JQ", job, tail+1)

Get a job:
   in("head", ? &head)
   out("head", head+1)
   in("JQ", ? &job, head)
Worker Process

```c
int min;
LINDA_BLOCK PATH;

worker()
    int hops, len, head;
    int path[MAXTOWNS];
    PATH.data = path;
    for (;;) {
        in("head", ?, &head)
        out("head", head+1)
        in("job", ?, &hops, ?, &len, ?, &PATH, head)
        tsp(hops, len, path); /* sequential TSP */
    }
```

TSP

```c
tsp(int hops, int len, int path[])
    int e,me;
    rd("minimum", ?, &min); /* update min */
    if (len ! min)
        if (hops == (NRTOWNS-1))
            in("minimum", ?, &min);
            min = minimum(len,min);
            out("minimum",min);
        else
            me = path[hops];
            for (e=0; e < NRTOWNS; e++)
                if (!present(e,hops,path))
                    path[hops+1] = e;
                    tsp(hops+1,len+distance[me][e], path);
```
**Master**

```c
master(int hops, int len, int path[])  
    int e, me;  
    if (hops == MAXHOPS)  
        PATH.size = hops + 1; PATH.data = path;  
        out("tail", &tail)  
        out("tail", tail+1)  
        out("job", hops, len, PATH, tail+1)  
    else  
        me = path[hops];  
        for (e=0; e < NRTOWNS; e++)  
            if (!present(e, hops, path))  
                path[hops+1] = e;  
                master(hops+1, len+distance[me][e], path);  
```

**Discussion**

- Communication is "uncoupled" from processes
- The model is machine-independent
- The model is very simple
- Possible efficiency problems
  - Associative addressing
  - Distribution of tuples
Implementation of Linda

- Linda has been implemented on
  - Shared-memory multiprocessors (Encore, Sequent, VU Tadpole)
  - Distributed-memory machines (S/Net, workstations on Ethernet)

- Main problems in the implementation
  - Avoid exhaustive search (optimize associative addressing)
  - Potential lack of shared memory (optimize communication)

Components of Linda Implementation

- Linda preprocessor
  - Analyzes all operations on Tuple Space in the program
  - Decides how to implement each tuple
- Runtime kernel
  - Runtime routines for implementing TS operations
Linda Preprocessor

Partition all TS operations in disjoint sets
Tuples produced/consumed by one set cannot be
produced/consumed by operations in other sets

OUT("hello", 12);

will never match

IN("hello", 14); constants don't match
IN("hello", 12, 4); number of arguments doesn't match
IN("hello", ? &aFloat); types don't match

Classify Partitions

- Based on usage patterns in entire program
- Use most efficient data structure
  - Queue
  - Hash table
  - Private hash table
  - List
Case 1: Queue

OUT("foo", i);
IN("foo", ? &j);
First field is always constant => can be removed by
the compiler
Second field of IN is always formal => no runtime
matching required

Case 2: Hash Tables

OUT("vector", i, j);
IN("vector", k, ? &l);
First and third field same as in previous example
Second field requires runtime matching
Second field always is actual => use hash table end
Case 3: Private Hash Tables

OUT("element", i, j);
IN("element", k, &j);
RD("element", &k, &j);
Second field is sometimes formal, sometimes actual
If actual => use hashing
If formal => search (private) hash table

Case 4: Exhaustive Search in a List

OUT("element", &j);
Only occurs if OUT has a formal argument => use exhaustive search
Runtime Kernels

- Shared-memory kernels
  - Store Tuple Space data structures in shared memory
  - Protect them with locks
- Distributed-memory (network) kernels
  - How to represent Tuple Space?
  - Several alternatives:
    - Hash-based schemes
    - Uniform distribution schemes

Hash-based Distributions

- Each tuple is stored on a single machine, as determined by a hash function on:
  1. search key field (if it exists), or
  2. class number
- Most interactions involve 3 machines
  P1: OUT(t); => send t to P3
  P2: IN(t); => get t from P3
Uniform Distributions

- Network with reliable broadcasting (S/Net)
  broadcast all OUTs => replicate entire Tuple Space
  RD done locally
  IN: find tuple locally, then broadcast
- Network without reliable broadcasting (Ethernet)
  OUT is done locally
  To find tuple (IN/RD), repeatedly broadcast

Performance of Tuple Space

- Performance is hard to predict, depends on
  - Implementation strategy
  - Application
- Example: global bound in TSP
  - Value is read (say) 1,000,000 times and changed 10 times
  - Replicated Tuple Space => 10 broadcasts
  - Other strategies => 1,000,000 messages
- Multiple TS operations needed for 1 logical operation
  - Enqueue and dequeue on shared queue each take 3 operations
Conclusions on Linda

- Very simple model
- Distributed data structures
  Can be used with any existing base language

- Tuple space operations are low-level
- Implementation is complicated
- Performance is hard to predict

Reference

