Linda and Its Tuple Spaces

Abhinav S Bhatele (CS498lvk)

Introduction

- It is essentially a shared-memory programming model based on tuple-spaces
- Only cares about process creation and co-ordination
- How and what the process computes is a black box to the model
- A base language with the addition of the tuple operations yields a parallel programming ‘dialect’

Tuple Spaces

- A tuple is a series of typed fields, for example

```
("a string", 15.01, 17, "another string")
```

- Processes share a tuple space between them which has tuples floating in it
- A tuple can be a simple 'data' tuple which can be read and/or removed
- It can be a 'live' tuple which carries out some computation of its own

The model

- It is based on generative communication
- A process wishing to send data to another creates a tuple and sets it adrift in the tuple space
- A process looking for data tries to match tuples in the tuple space with its own
- If a new process is required for a computation, the parent process releases a live tuple in the tuple space

Tuple Operations

- To create/ send tuples
  - out – send a data tuple into the tuple space
  - eval – send a live tuple into the tuple space which gets evaluated into a ‘data’ tuple
- To read/ receive tuples
  - in – read and remove a tuple from the tuple space
  - rd – simply read a tuple from the tuple space

Implications of this model

- Communication and process creation are two facets of the same operation
- Data is exchanged in the form of persistent objects and not transient messages
- It promotes an uncoupled programming style – the senders and receivers need not know about each other

Examples

- Matching a tuple to get data –

```
out(“a string”, 15.01, 17, “another string”)
in(“a string”, ? f, ? i, “another string”)
```

- Creating data structures out of tuples –

```
(“V”, 1, FirstElt)
(“V”, 2, SecondElt)
(“V”, 3, ThirdElt)
```

- Change the ith element –

```
in(“V”, i, ? OldVal)
out(“V”, i, NewVal)
```

Dining Philosopher's Problem

```
phil(i)
   int i;
{
   while(l) {
   think();
   in("room ticket");
   in("chopstick", i);
   in("chopstick", (i+l)%Num);
   eat();
   out("chopstick", i);
   out("chopstick", (i+i)%Num);
   out("room ticket");
   }
}
```

Server-Clients

```
server()
{
    int index = 1;
    . . .
    while(1) {
         in("request", index, ? req);
         . . .
         out("response", index++, response);
    }
}

client()
{
    int index;
    . . .
    in("server index", ? index);
    out("server index", index+l);
    . . .
    out("request", index, request);
    in("response", index, ? response);
    . . .
}
```

References

1. Sudhir Ahuja, Nicholas Carriero and David Gelernter, *Linda and Friends*, IEEE Computer, Aug. 1986
2. Nicholas Carriero and David Gelernter, *Linda in Context*, Communications of the ACM, Vol. 32, No. 4, April 1989
3. L. V. Kale, *Technical Correspondence on Linda in Context,* Communications of the ACM, Vol. 32, No. 10, Oct. 1989, pp. 1252-1253.
4. Nicholas Carriero and David Gelernter, *How to Write Parallel Programs: A Guide to the Perplexed*, ACM Computing Surveys, Vol. 21, No. 3, Sept. 1989
5. Nicholas Carriero and David Gelernter, *How to Write Parallel Programs: A First Course*


Thank you!
teamwork