Outline

- Motivation
- Linda Tuple Spaces
- Modelling Tuple Spaces in FSP
- Implementing Tuple Spaces in Java
- Supervisor-Worker Model
- Supervisor-Worker Java Implementation
Motivation

- Exploiting parallel execution on multiple processors
- Communication between different processors by a connector called “bag”
  - Supervisor creates tasks and puts them into bag
  - Workers pick tasks from bag and perform them
- Workers may themselves be supervisors

Supervisor-Worker Architecture
Linda Tuple Spaces

- **Primitive for implementing “bag” connectors.**
- **Tuple is a tagged data record:**
  - Tuples are exchanged in tuple spaces using associative memory.
- **Available basic operations:**
  - `out(“tag”, expr1, …, exprn)`
  - `in(“tag”, field1, …, fieldn)`
  - `rd(“tag”, field1, …, fieldn)`
  - `inp(“tag”, field1, …, fieldn)`
  - `rdp(“tag”, field1, …, fieldn)`

Linda Basic Operations

- `out(“tag”, expr1, …, exprn)`
  - Execution completes when the expressions have been evaluated and the resulting tuple deposited in the tuple space.
- `in(“tag”, field1, …, fieldn)`
  - Execution blocks until the tuple space contains a tuple matching the specified fields.
- `rd(“tag”, field1, …, fieldn)`
  - Like `in`, but doesn’t remove tuple from tuple space.
- `inp(“tag”, field1, …, fieldn)`
- `rdp(“tag”, field1, …, fieldn)`
  - Non-blocking versions of `in` and `rd`
Linda in operation

\[ \text{in}(\text{"tag"}, \text{field}_1, ... , \text{field}_n) \]

- **fields** are either:
  - the name of a local variable in the process calling \text{in}
  - an expression to evaluate

A tuple in tuple-space matches the \text{in} request if:

- the tag is identical
- the number of fields is the same
- the expressions equal the values in the tuple.
- the variables have the same type as the values in the tuple.

---

Tuple Space Model

\[
\begin{align*}
\text{const N=2} \\
\text{set Tuples=\{any\}} \\
\text{const False} = 0 \\
\text{const True} = 1 \\
\text{range Bool} = \text{False..True}
\end{align*}
\]

\[
\text{TUPLE}(T='\text{any}) = \text{TUPLE}[0], \\
\text{TUPLE}[i:0..N]=\begin{cases}
\text{when } (i<N) \text{ out}[T] & \rightarrow \text{TUPLE}[i+1] \\
\text{when } (i>0) \text{ in}[T] & \rightarrow \text{TUPLE}[i-1] \\
\text{when } (i>0) \text{ inp[True]}[T] & \rightarrow \text{TUPLE}[i-1] \\
\text{when } (i==0)\text{ inp[False]}[T] & \rightarrow \text{TUPLE}[i] \\
\text{when } (i>0) \text{ rd}[T] & \rightarrow \text{TUPLE}[i] \\
\text{rdp[i>0]}[T] & \rightarrow \text{TUPLE}[i].
\end{cases}
\]

\[ ||\text{TUPLESPACE} = \text{forall } [t:Tuples] \text{TUPLE}(t). \]
Tuple Model LTS

<table>
<thead>
<tr>
<th>-1</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>inp.0.any</td>
<td>out.any</td>
<td>rdp.0.any</td>
<td>rd.any</td>
</tr>
<tr>
<td>inp.1.any</td>
<td>inp.any</td>
<td>rdp.1.any</td>
<td>rdp.any</td>
</tr>
<tr>
<td>out.any</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: in these action names, 0 corresponds to false, 1 corresponds to true.

Tuple Space Java Implementation

```java
public interface TupleSpace {
    // deposits data in tuple space
    public void out(String tag, Object data);
    // extracts object with tag from tuple space
    public Object in(String tag) throws InterruptedException;
    // reads object with tag from tuple space
    public Object rd(String tag) throws InterruptedException;
    // extracts object if avail else return null
    public Object inp(String tag);
    // read object if avail else return null
    public Object rdp(String tag);
}
```
Supervisor-Worker Algorithm

- **Supervisor:**
  
  forall tasks do out("task",...)
  forall results do in("result",...)
  out("stop")

- **Worker:**
  
  while not rdp("stop") do
  in("task",...)
  compute result
  out("result",...)

Supervisor-Worker Model

Need a maximum on duplicate tuples:

const N = 2

Three tuple types:

set Tuples = {task,result,stop}

Tuple alphabet:

set TupleAlpha =
{{in,out,rd,rdp[Bool],inp[Bool]}.Tuples}

Supervisor outputs tasks, inputs results, and then signals the workers to terminate:

SUPERVISOR = TASK[1],
TASK[i:1..N] = (out.task ->
  if i<N then TASK[i+1] else RESULT[1]),
RESULT[i:1..N] = (in.result ->
  if i<N then RESULT[i+1] else FINISH),
FINISH = (out.stop->end->STOP)+TupleAlpha.
Supervisor-Worker Model

Worker checks if it needs to stop, otherwise inputs task, outputs results:

WORKER = (rdp[b:Bool].stop->
if (!b) then (in.task->out.result->WORKER)
else (end -> STOP ) + TupleAlpha.

Hack to avoid spurious deadlock detection:
END = (end -> ENDED), ENDED = (ended-> ENDED).

Glue it all together:
||SUPERVISOR_WORKER={(supervisor: SUPERVISOR
|||{redWork, blueWork}: WORKER
|||{supervisor, redWork, blueWork}: TUPLESPACE
|||END)
/{end/{supervisor, redWork, blueWork}.end}.

Supervisor and Worker LTS

SUPERVISOR

WORKER

out.task → out.task → in.result → in.result → out.stop → end

rdp.0.stop → in.task → rdp.1.stop

out.result → end

© Wolfgang Emmerich, Mark Handley 1998 - 2003
Analysis of Supervisor-Worker Model

- **Trace to DEADLOCK:**
  ```
  supervisor.out.task
  supervisor.out.task
  redWork.rdp.0.stop
  redWork.in.task
  redWork.out.result
  supervisor.in.result
  redWork.rdp.0.stop
  redWork.in.task
  redWork.out.result
  supervisor.in.result
  redWork.rdp.0.stop
  supervisor.out.stop
  ```

  **Supervisor only outputs stop after red worker tries to read it.**
  **Red is waiting for a new task that never arrives.**

Deadlock Free Algorithm

- **Supervisor:**
  ```
  forall tasks do out("task",...)
  forall results do in("result",...)
  out("task", stop)
  ```

- **Worker:**
  ```
  while true do
    in("task",...)
    If value is stop
      then out("task",stop); exit
    compute result
    out("result",...)
  ```
Deadlock Free Model

set Tuples = \{task, task.stop, result\}

Supervisor as before, except different stop method:

SUPERVISOR = TASK[1],
TASK[i:1..N] = (out.task ->
    if i<N then TASK[i+1] else RESULT[1]),
RESULT[i:1..N] = (in.result ->
    if i<N then RESULT[i+1] else FINISH),
FINISH=(out.task.stop->end->STOP)+TupleAlpha.

Worker inputs task.stop and re-emits it for other workers:

WORKER=(in.task -> out.result -> WORKER
   | in.task.stop->out.task.stop->end->STOP
 )+ TupleAlpha.

Check for proper termination:

progress TERMINATION={ended}

Supervisor-Worker Example

Goal: Compute the area under a curve

- Approximate using rectangles
- Parallelize task by delegating computation of different rectangles to one of 4 workers
- Supervisor adds results computed by 4 workers
Supervisor-Worker Example Design

Summary

- **Motivation**
- **Linda Tuple Spaces**
- **Modelling Tuple Spaces in FSP**
- **Implementing Tuple Spaces in Java**
- **Supervisor-Worker Model**
- **Supervisor-Worker Java Implementation**