parmod detailed

Slides derived from a presentation
by Marco Danelutto

parmod overall

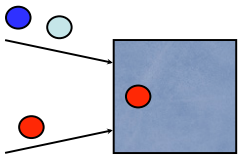
- process (multiple) input stream(s) of data
- produce (multiple) output stream(s)
- Streams
  - data flow semantics (sort of one way comms)

parmod Interface

- Parallel Computation
  - **Virtual Processes** (VP) express computation grain
  - VP eventually map to physical resources (automatic!)
- Parmod *minimal* syntax / semantics
  - bring data to VPs
  - define how VP cooperate
  - bring results out

Non-deterministic input control

Multiple data-flow inputs:
(how) do we choose?



- boolean guards
  - accessible and modifiable
- priorities
- input guards
- data availability

when satisfied, trigger virtual process(es)

Nondeterminism: input section

- non deterministic input control
  - set of data-flow input streams to choose from
- input section handles:
  - Priorities
  - Boolean Guards (enable input streams on expression)
  - Stream combinations (*f* needs both *A* and *B* to compute…)
- data from streams is *distributed* to
  - virtual processes or parmod state
  - **Distributions**: broadcast, unicast, scatter, multicast
- data availabilty triggers virtual processor execution (à la Data Flow)

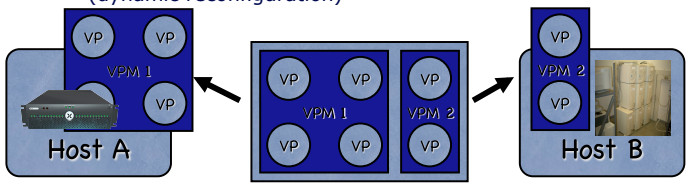
VP : logically parallel activity

- Concept of virtual process:
  - a logically concurrent/parallel activity
  - with a name
    - there is a topology arranging VPs
    - topology can be exploited to define the computation
  - can perform different functions
    - selects according to its state and inputs
    - sequential code modules encapsulated in a **proc**
- Computation is described in terms of code & data dependencies
  - VP possibly sharing state with the other activities

VP : logical and actual machines

At execution time:

- VP mapped to Virtual Processes Manager (VPM)
- VPM mapped to physical processing resources
  - Mapping performed by tools
  - Mapping can change at run-time (dynamic reconfiguration)



VP naming

- Topology = VP naming scheme
  - array: topology array [i:N] myVP;
    - processors name after indexes of a (multidimensional) array
    - topology array [i:N] [j:M] [k:O] myVP
  - none: topology none myVP;
    - none= no naming, anonymous processes (task farm)
    - can still express many different computation schemes
  - one: topology one myVP;
    - one single (seq) process, but all parmod features
    - e.g. multiple in/out, non deterministic input control

Parmod internal state

- attributes = variables (typed, structured)
- can be logically distributed on VPs
  - match attribute structure on parmod's topology
- owner-computes rule
- compiler + run time support ensure (safe) accessibility
- implemented through AdHOC
  - independent shared-memory support

Parmod distributions

- state to VPs
- input data to VPs and state
- scatter, broadcast, multicast + scheduled
- scheduled
  - computed on the basis of the input data

Parmod application code

- associated to virtual processes
  - to all or to subsets (using naming)
- Call through the *proc* code in C, C++, F77
  - Java soon ...
- possibility to introduce parmod iterations
  - for, while statements
- Input data triggers code execution
- Barriers can be automatically inserted
  - take care of data-parallel synchronizations

Parmod output section

- Simple syntax for simple cases
  - output parameters of virtual processes simply delivered to output streams
- User control for more complex cases
  - assist_out(stream, object)
  - recompose data structures out of VP results
  - insert arbitrary proc (attributes, guards)
- Multiple output stream handling

external objects

run time code access to invoke external services

e.g. CCM, WS, AdHOC, shared objects, ...

proc code can access these services under complete user control

sort of ESCape to structured parallelism ...

Examples of structured patterns

(parmod subcases)

- task farm
  - topology none, distribution on-demand, collect from any
- "dedicated" task farm
  - topology array, distribution scheduled
- (embarrassingly) data parallel
  - topology array, tree
- fixed/variable stencil data parallel
  - topology array, tree
- Custom schemes
  - topology array, tree + non det input section+ state + multiple VP proc + code within output section

Parmod Examples

Check parmod examples at

http://www.cli.di.unipi.it/doku/doku.php/magistraleinformaticanetworking/spd/