The ASSIST parallel component model

Marco Danelutto
Dept. Computer Science - Univ. Pisa
The context

- Component based programming
- code reuse, deployment, business model ...
- Parallelism within the single component
- high performance “services”
- transparently provided to the user
The thesis

- classical parallel programming models are too low level
  - message passing, shared memory (mechanisms)
  - SPMD, OO (models)

- the level of abstraction should be raised (beyond the "physical machine" concepts)
The implementation

ASSIST (A Software development System based on Integrated Skeleton Technology)

- structured parallelism exploitation
- targeting COW -> NOW -> GRID
- encapsulated in components
- developed within the 3 year GRID.it national project (ends in 2006)
Outline of the talk

- the ASSIST parmod
- the components
- the experimental results
- the hints and suggestions to Task 3.1
- the hints and suggestions to Task 3.2/3
the parmod

parmod = generic PARallel MODule

structured way of exploiting parallel computations

abstracting:

logical parallel activities

logical data sharing

from the actual mechanisms needed
The ASSIST parallel component model

M. Danelutto, Dept. Computer Science, Univ. of Pisa

Programming models and components for the grid workshop - Sophia Antipolis - Oct 14, 2005 GRID@works

shared state

user def virtual processes

non det control on input

external object access

multiple output streams

The picture

VP VP VP

VP VP VP

VP VP VP
I/O

- parmod processes (multiple) input stream data
- to produce (multiple) output stream data
- streams
- data flow semantics
- (sort of one way comms)
non deterministic input control

- set of data flow input streams
- handled in input section
- data from streams to
  - virtual processes or parmod state (broadcast, unicast, scatter, multicast)
- data availability triggers virtual processor execution (à la Data Flow)
non deterministic input control

+ boolean guards (accessible and modifiable)
+ priorities
+ data availability

= input guard

 occasio when satisfied triggers virtual process(es)
logical parallel activities

- concept of virtual process:
  - a logically concurrent/parallel activity
  - with a name
  - described in terms of code & data dependencies
  - possibly sharing state with the other activities
logical vs. machines

- Virtual Processes (VP) mapped to Virtual Processes Manager (VPM)
- VPM mapped to physical processing resources
naming

- topology
  - vector, array: topology array \([i:N]\) myVP;
  - process name after indexes
  - none: topology none myVP;
  - anonymous processes (task farm)
  - one: topology one myVP;
  - seq process with non det input control
internal state

- just variables
- logically distributed on virtual processes
- owner computes rules
- compiler + run time support ensure (safe) accessibility
- implemented through SMU/AdHOC independent run time support
distributions

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

- associated to virtual processes
- to all or to subsets (using naming)
- call to seq code (C, C++, F77; Java soon ...)
- possibility to introduce iterations
- code execution triggered by input data availability
- implicit barriers for synchronizations
output

- In simple cases, output parameters of virtual processes simply delivered to output streams
- User control for more complex cases
- assist_out(stream, object)
- 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 ...
Structured patterns
(parmod subcases)

- task farm (topology none)
- "dedicated" task farm (topology array scheduled)
- (embarrassingly) data parallel (topology array, tree)
- (fixed, variable) stencil data parallel (topology array, tree)
- custom (topology array, tree + non det input + state + diff virtual processes sets + output)

"The ASSIST parallel component model" M. Danelutto, Dept. Computer Science, Univ. of Pisa
Programming models and components for the grid workshop - Sophia Antipolis - Oct 14, 2005 GRID@works
Component (the basics)

- parmod automatically wrapped to
  - CCM
  - WS
- compile flag matter!
- RPC semantics: fixed name method with type
  Type Input Stream $\rightarrow$ Type OutputStream
Component (the news)

- Autonomic control + approximate performance model
- Adaptivity to target GRID features
- Exploits structuring
- Ensures (best effort) user defined performance contracts

Whole ASSIST programs as components (arbitrary graphs of parmods and seqs)
Autonomic control

“The ASSIST parallel component model” M. Danelutto, Dept. Computer Science, Univ. of Pisa
Programming models and components for the grid workshop – Sophia Antipolis – Oct 14, 2005 GRID@works
Autonomic control (2)

[Graph showing autonomic control metrics over time]

N. of VPMs in parmod
VPMs aggregated power
QoS contract
Input stream queue fill level

Wall Clock Time (s)
Component (the GRID.it way)

- functional / non functional features
- use/provide, events, streams
- autonomic control
- supercomponents
  - managed, structured component composition
- task farm, generic graph/pipeline

“The ASSIST parallel component model” M. Danelutto, Dept. Computer Science, Univ. of Pisa
Programming models and components for the grid workshop – Sophia Antipolis – Oct 14, 2005 GRID@works
Autonomic control (the GRID.it way)

"The ASSIST parallel component model" M. Danelutto, Dept. Computer Science, Univ. of Pisa
Programming models and components for the grid workshop - Sophia Antipolis - Oct 14, 2005 GRID@works
hints & suggestions (3.1)

- SPMD is not sufficient
- OO difficult to optimize in the run time
- structured parallelism exploitation helps
- to achieve efficiency
- to relieve programmers from burden
- autonomic control is fundamental for grids
hints & suggestions (3.2/3)

- structured compositions as first class citizens
- application managers (hierarchical autonomic control) to take care of grid peculiarities
- streams to enhance composition mechanisms
- advanced GCM → higher level than middleware
any questions?

FirstName.Name@di.unipi.it

Corrado.Zoccolo ASSIST in general
Marco.Aldinucci adaptivity
Massimo.Coppola CCM
Pietro.Vitale installation

www.di.unipi.it/Assist.html

www.grid.it
parmod elabora (input_stream T_cart A[N]
output_stream T_cart B[N]) {

topology array [i:N] Pv;
attribute long S[N] scatter S[*u] onto Pv [u];
stream long ris;

do input_section {
  guard1: on , , A {
    distribution A[*v] scatter to S[v];
  }
} while (true)

distribution A[*v] scatter to S[v];
} while (true)

virtual_processors {
  elab (in guard1 out ris) {
    VP i {
      for (h=0;h<N;h++) {
        Felab (in S[i] out S[i]);
      }
      assist_out(ris, S[i]);
    }
  }
}

output_section {
  collects ris from ALL Pv[i] {
    int el;
    int B_[N];
    AST_FOR_EACH(el) {
      B_[i]=el;
      assist_out(B,B_);
    }
  }
}<>;
}
parmod elabora (input_stream T_cart A[N] output_stream T_cart B[N])
{
    topology none Pv;

    do input_section {
        guard1: on , , A {
            distribution A on_demand to Pv;
        }
    } while (true)

    virtual_processors {
        elab (in guard1 out B) {
            VP {
                Felab (in A out B);
            }
        }
    }

    output_section {
        collects B from ANY Pv;
    }
}
parmod code sample (3)

parmod p_none (input_stream long A1,long A2
output_stream long B1[N], long B2[N]) {

topology none Pv;

do input_section {
  guard1: on , , A1 {
    distribution A1 broadcast to Pv;
  }

  guard2: on , , A2 {
    distribution A2 on_demand to Pv;
  }
} while (true)

virtual_processors {
  elab1 (in guard1 out B1) {
    VP {
      f_p1 (in A1 output_stream B1);
    }
  }

  elab2 (in guard2 out B2) {
    VP {
      f_p2 (in A2 output_stream B2);
    }
  }
}

output_section {
  collects B1 from ANY Pv;
  collects B2 from ANY Pv;
}
```
parmod p_array(input_stream long A1, long A2
output_stream long B1[N],
long B2[N]) {

  topology array [i:N] Pv;

  do input_section {
    guard1: on , , A1 {
      distribution A1 broadcast to Pv;
    }
    guard2: on , , A2 {
      distribution A2 broadcast to Pv;
    }
  } while (true)
}

virtual_processors {
  elab1(in guard1 out B1) {
    VP i {
      f_p1(in A1 output_stream B1);
    }
  }
  elab2(in guard2 out B2) {
    VP i {
      f_p2(in A2 output_stream B2);
    }
  }
}

output_section {
  collects B1 from ANY Pv;
  collects B2 from ANY Pv;
}
```

“The ASSIST parallel component model”  M. Danelutto, Dept. Computer Science, Univ. of Pisa Programming models and components for the grid workshop - Sophia Antipolis - Oct 14, 2005 GRID@works
Autonomic control

“The ASSIST parallel component model” M. Danelutto, Dept. Computer Science, Univ. of Pisa
Programming models and components for the grid workshop – Sophia Antipolis – Oct 14, 2005 GRID@works