Efficient Composition of PVM Programs

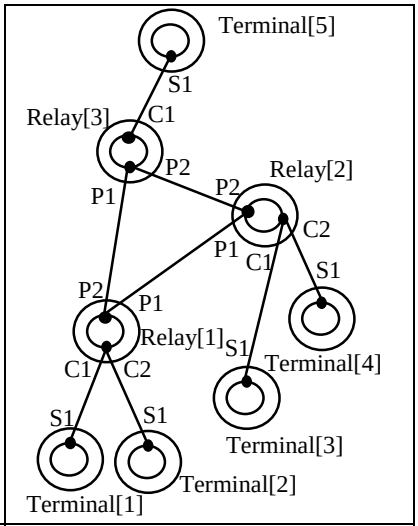
J.Y. Cotronis, E. Floros, N. Papazis
Dept. of Informatics, Univ. of Athens, Panepistimiopolis, 157 71 Athens, GREECE.
tel.: +30 1 7230172 fax: +30 1 7219 561 e-mail: [cotronis, floros, papazis]@di.uoa.gr

Overview

Programs forming tree process communication topologies, each process communicating only with its parent and children processes, are well suited to PVM [2], whilst programs forming graph topologies demand complex programming. We outline a methodology for the efficient composition of PVM programs; we use as an example the Distribution of Maximum application: there are *terminal* processes which hold an integer value and require the maximum of their values; each terminal process sends its value to an associated *relay* process and (eventually) receives from it the required maximum; relays receive values from their terminals, find their local maximum LM, exchange their LMs with the other relays, find their maximum, and they send it to their terminal processes. The implementation (three relays, five terminals)

```
APPLICATION Distribution_Maximum;
PCG
Components
  terminal port-types: S;
  relay     port-types: C, P;
Processes
  relay[1], relay[2]              #ports= C:2, P:2;
  relay[3]                        #ports= C:1, P:2;
  terminal[1], terminal[2], terminal[3],
  terminal[4], terminal[5]        #ports= S:1;
Channels
  terminal[1].S[1] ?   relay[1].C[1];
  terminal[2].S[1] ?   relay[1].C[2];
  terminal[3].S[1] ?   relay[2].C[1];
  terminal[4].S[1] ?   relay[2].C[2];
  terminal[5].S[1] ?   relay[3].C[1];
  relay[1].P[1] ?    relay[2].P[1];
  relay[1].P[2] ?    relay[3].P[1];
  relay[2].P[2] ?    relay[3].P[2];
```

PCG Builder




```
PARALLEL SYSTEM
Environment PVM3
PVM3_Annotation
  tagID : default;
PVM3_Options
 Allocation
  relay[1], terminal[1], terminal[2] at euridiki;
  relay[2], terminal[3], terminal[4] at kadmos;
  relay[3], terminal[5]              at lavdakos;
SEQUENTIAL COMPONENTS
Executable files
  terminal : path default file terminal;
  relay    : path default file relay;
Execution Parameters
  terminal[1]: 6;  terminal[2]: 999;
  terminal[3]: 7;  terminal[4]: 8;  terminal[5]:9;
```

The application script

annotation B.

```
Node 1  name        : relay[1]
        allocation  : euridiki
        file        : relay
        path        : Default
        parameters: (None)
Node 4  name        : terminal[1]
        allocation  : euridiki
        file        : terminal
        path        : Default
        parameters: 6

Channel 1 :4.S[1] ?   1.C[1] tagid 1
Channel 2 :5.S[1] ?   1.C[2] tagid 2
Channel 6 :1.P[1] ?   2.P[1] tagid 6
Channel 7 :1.P[2] ?   3.P[1] tagid 7
Channel 8 :2.P[2] ?   3.P[2] tagid 8
```

The annotated PCG

is outlined, depicting the three facets of the methodology:

1. The annotated Process Communication Graph (PCG) is produced from a script having three main parts: The first, headed by PCG, specifies the general PCG, produced by the PCG-builder. Components define port types for process interconnection; a process, an instantiation of a reusable program component (see 2), may have a number of ports of the same type. The second, headed by Parallel System, specifies the annotation of nodes (processes) and arcs (channels) with information required for the creation and interconnection of processes. The third, headed by Sequential Components, specifies process loading information (executables and parameters). The annotation Builder annotates the PCG. The script of the application, its PCG and the annotation of some of its nodes and arcs are shown above.

2. The reusable program components do not assume any specific communication structure for their instantiated processes but specify open communication interfaces. For point-to-point communication between two PVM processes two values are needed: the tid of the other process and a common tagid. Components may have a number of ports, pairs (tid, tagid), of each type stored in array Interface. Each port is referred by its type and a port index within the type. The components have the same

```
void main(Int)                          /* terminal*/
{ InterfaceType Interface[1];
  MakePorts(Interface);    /*Loader Phase  1 */
  SetInterface(Interface);  /* Loader Phase 2 */
  realmain(Interface);       /* main action */      }
void realmain (Interface);       { terminal code }
```

The structure of terminal component

```
void main()                          /* relay */
{ InterfaceType  Interface[2];
  MakePorts(Interface);
  SetInterface(Interface);
  realmain(Interface);                          }
void realmain(Interface); { relay code }
```

The structure of relay component

structure (above): they declare array Interface, its size being the number of port types, they call MakePorts to receive from the Loader (see 3) information to create the appropriate number of ports, then call SetInteface to receive from the Loader and set the (tid,tag) port pairs and they call their realmain actions.

3. The universal PVM Loader visits the annotated PCG nodes and spawns processes as its children-processes; all required information annotates the nodes: the executables, process allocation and the command line parameters. The Loader sends to each spawned process the number of its ports of each type (received by MakePorts of the process). The tid of the process ports is generally not known as processes may have not been spawned yet. The Loader annotates the node with the tid of the spawned process. Having spawned all processes, the Loader visits again the nodes and sends to each process its port information, i.e. (tid, tagid) pairs (received by SetInterface of the process). The parallel program is now running.

By editing the scripts we may scale programs (adding terminals and relays), specify the host allocation of processes or even modify the topology of the processes, without modifying the components, which are reusable as library components [1].

References

[1] J.Y.Cotronis: Efficient composition and automatic initialization of arbitrary structured PVM programs, IFIP Proceedings of the 1st Workshop on Soft. Engin. for Parallel and Distr. Systems (in association with ICSE 96, Berlin, March 96)

[2] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, Vaidy Sunderam, 'PVM 3 User's guide and Reference Manual', ORNL/TM-12187, May 1994.