

//ELLPACK Library: Parallel Templates

OVERVIEW

The templates listed in this chapter describe how to get performance trace data for the execution of parallel programs using the PICL and PARMACS communication libraries. This trace data can then be used with state-of-the-art performance analysis visualization and animation programs.

All of the //ELLPACK parallel modules have been ported to the PICL and PARMACS communication libraries, and are available on a variety of parallel platforms.

TEMPLATES

The parallel templates described in this chapter are:

PARMACS TRACE

PICL TRACE

PARMACS TRACE

//ELLPACK INTERFACE

Sang-Bae Kim and Shahani Markus

PURPOSE

The Parmacs Trace is used for understanding the run time behavior of the application from which it has been generated. It contains information that describe the parallel activity of the application, e.g. message passing features and execution times. Furthermore, using this information, observations can be made and conclusions can be drawn regarding the performance of the //ELLPACK application.

FUNCTIONALITY

By enabling the generation of a PARMACS Trace from a //ELLPACK application, the user automatically obtains a *run-time "identity"* of the application in terms of communication and computation events. These events, along with additional information, are collected in the PARMACS Trace. When the application has finished, the user can introduce some other tools to perform an analysis of the application in a *postmortem* fashion.

Setting up a //ELLPACK application to produce a PARMACS Trace causes every single communication event to be recorded and stored in a file. This file is called **Trace**.

//ELLPACK USAGE

In order to use (produce) the PARMACS Trace the following steps must be followed:

- The appropriate makefile must be specified when generating the //ELLPACK program for the application.
- The appropriate declaration part must be added in the text specified within the Session Editor.

.....

INTERFACE

The Interfacing consists of two steps:

(1) in order to generate the //ELLPACK application, use the makefile as below:

```
~/ELLPACK\  
-k -t -i -w -u tmp2 -m makefile.ncube2.pm <filename>
```

(2) add the following declaration section in the text described within the Session Editor:

```
DECLARATIONS. +host./* PARMACS Graph Mapping*/  
common / c9mapping / mapping  
character*5 mapping  
data mapping /'graph'/
```

In case the **Torus** mapping is the one needed, simply change the graph in the above with torus.

RESTRICTIONS

Several restrictions govern the use of the PARMACS Trace. Some of them are listed below:

- The number of nodes for which a trace can be exploited by other tools is limited in most cases to sixteen.
- The size of the trace file generated for applications with substantial communication activity is prohibitively large.
- The performance degradation of the application, especially when the number of nodes is more than sixteen should be considered substantial.
- It is the case that to generate a large trace file requires considerably long time.

PERFORMANCE ESTIMATES

In general, when the //ELLPACK has been instructed to generate a PARMACS Trace, the overall execution time is increased. The overhead on the execution that might be experienced increases with the

.....

number of nodes, as someone would expect. Elaborate experimentation has resulted in a consolidated table of the expected overhead [cf. 2].

ACKNOWLEDGEMENTS

REFERENCES

Pallas GmbH, *PARMACS, PA-TOOLS*, June 1992.

EXAMPLE

Below is an example of the shell script command for compiling a // ELLPACK application along with the text of the Session Editor for that application, set up to produce a PICL trace.

```
/* Shell script portion for compiling the application */

#! /bin/csh -f

setenv //ELLPACK /usr//ELLPACK/current

set echo
/usr//ELLPACK/current/bin/sun4//ELLPACK \
    -k -t -i -w -u tmp2 -m makefile.ncube2.pm <filename>
/* Text as appears in the Session Editor */

OPTIONS.
    clockwise =.true.
    xplot3d

DECLARATIONS. +both. +boundary.
    common / plotx / xc(6, 8)
    common / ploty / yc(6, 8)

DECLARATIONS. +host.
    common / c9mapping / mapping
    character*5 mapping
    data mapping /'graph'/

FORTRAN. +both. +boundary.
```

.....

```
data (xc(1,j),j=1,8) / 0.06,0.06,0.0,0.0,0.0,0.0,0.0,0.0 /
  data (xc(2,j),j=1,8) / 0.06,0.94,0.0,0.0,0.0,0.0,0.0,0.0 /
  data (xc(3,j),j=1,8) / 0.94,0.94,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (xc(4,j),j=1,8) / 0.94,0.72,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (xc(5,j),j=1,8) / 0.72,0.72,0.72,0.50,0.28,0.28,0.28,0.28 /
a,0.28,0.28 /
  data (xc(6,j),j=1,8) / 0.28,0.06,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (yc(1,j),j=1,8) / 0.06,0.94,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (yc(2,j),j=1,8) / 0.94,0.94,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (yc(3,j),j=1,8) / 0.94,0.06,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (yc(4,j),j=1,8) / 0.06,0.06,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
  data (yc(5,j),j=1,8) / 0.06,0.28,0.50,0.50,0.50,0.50,0.28,0.28 /
a,0.06,0.06 /
  data (yc(6,j),j=1,8) / 0.06,0.06,0.0,0.0,0.0,0.0,0.0,0.0 /
a,0.0,0.0 /
FOR t = 0.00 TO 1.00
```

EQUATION. UXX+2*UYY = FORCE(X,Y)

BOUNDARY. +parm.

```
U = true(x,y) ON X = paramlx(1,t,2,6), &
Y = paramly(1,t,2,6) &
FOR t = 0.00 TO 1.00
U = true(x,y) ON X = paramlx(2,t,2,6), &
Y = paramly(2,t,2,6) &
FOR t = 0.00 TO 1.00
U = true(x,y) ON X = paramlx(3,t,2,6), &
Y = paramly(3,t,2,6) &
FOR t = 0.00 TO 1.00
U = true(x,y) ON X = paramlx(4,t,2,6), &
Y = paramly(4,t,2,6) &
FOR t = 0.00 TO 1.00
U = true(x,y) ON X = paramlx(5,t,8,6), &
Y = paramly(5,t,8,6) &
FOR t = 0.00 TO 1.00
U = true(x,y) ON X = paramlx(6,t,2,6), &
Y = paramly(6,t,2,6) &
FOR t = 0.00 TO 1.00
```

machine. number of pes = 4

decomposition.

```
read fem decomposition from file &
(filename='/picltmp/testlpicl.dec', &
i9ndom=4, &
illmnpt=201, ilmelm=334, illmnpt=65, illmelm=99)
```

```

. . . . .
DISCRETIZATION. Bi-Linear FEM
SOLUTION. Jacobi CG
OUTPUT. max (error)
SUBPROGRAMS. +both.
    function FORCE(x,y)
        FORCE = 15.0*(2*SQRT(Y)+SQRT(X))/4.0
        return
    end

    function true(x,y)
        TRUE = Y**2.5+X**2.5
        return
    end
END.

```

PICL TRACE

//ELLPACK INTERFACE

Sang-Bae Kim and Shahani Markus

PURPOSE

The PICL Trace is used for understanding the run-time behavior of the application for which it was generated. It contains information that describe the parallel activity of the application, e.g. message passing features and execution times. Furthermore, using this information, observations can be made and conclusions can be drawn regarding the performance of the //ELLPACK application.

FUNCTIONALITY

By enabling the generation of a PICL Trace from a //ELLPACK application, the user automatically obtains a run-time “identity” of the application in terms of communication and computation events. These events, along with additional information, are collected in the PICL Trace. When the application has finished, the user can introduce some other tools to perform an analysis of the application in a *post-mortem* fashion. Setting up a //ELLPACK application to produce a PICLTrace causes every single communication event to be recorded and

.....

stored in a file. This file is called Trace.

//ELLPACK USAGE

In order to use (produce) the PICL Trace the following steps must be followed:

- (1) The appropriate makefile must be specified when generating the //ELLPACK program for the application.
- (2) The appropriate declaration part must be added in the text specified within the Session Editor.

INTERFACE

The Interfacing consists of two steps:

- (1)

```
~//ELLPACK \  
-k -t -i -w -u tmp2 -m makefile.ncube2.pc <filename>
```

- (2) add the following declaration section in the text described within the Session Editor (add immediately after the OPTIONS segment) :

```
DECLARATIONS. +node.  
common / c9picl / i9sbuf, i9flsh, i9even, i9comp, i9comm  
data i9sbuf, i9flsh, i9even, i9comp, i9comm /10000,1, 4,4,4/
```

RESTRICTIONS

Several restrictions govern the use of the PICL Trace. Some of them are listed below:

- The number of nodes for which a trace can be exploited by other tools is limited in most cases to thirty two.
- The size of the trace file generated for applications with substantial communication activity is prohibitively large.

.....

- The performance degradation of the application, especially when the number of nodes is more than sixteen should be considered substantial.
- It is the case that to generate a large trace file requires considerably long time.

PERFORMANCE ESTIMATES

In general, when the //ELLPACK has been instructed to generate a PICL Trace, the overall execution time is increased. The overhead on the execution that might be experienced increases with the number of nodes, as someone would expect. Elaborate experimentation has resulted in a consolidated table of the expected overhead [cf. 2].

REFERENCES

Geist et. al., “*A User’s Guide to PICL--A Portable Instrumented Communications Library*”, February 1992.