Parallel Debugging

Matthias Müller, Pavel Neytchev, Rainer Keller, Bettina Krammer

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

Outline

- Motivation
- Approaches and Tools
  - Memory Tracing Tools
    - **Valgrind**
  - MPI-Analysis Tools
    - **Marmot**
  - Debuggers
    - **TotalView**
    - **DDT**

Motivation - Problems of Parallel Programming

- All problems of serial programming
- Additional problems:
  - Increased difficulty to verify correctness of program
  - Increased difficulty to debug N parallel processes
  - New parallel problems:
    - **deadlocks**
    - **race conditions**
    - **irreproducibility**

Tools and Techniques to Avoid and Remove Bugs

- Programming techniques
- Static Code analysis
  - Compiler (with –Wall flag or similar), lint
- Runtime analysis
  - Memory tracing tools
  - Special OpenMP tools (assure, thread checker)
  - Special MPI tools
- Post mortem analysis
  - Debuggers

skipped

What is a Debugger?

- Common Misconception:
  A debugger is a tool to find and remove bugs
- A debugger does:
  - tell you where the program crashed
  - help to gain a better understanding of the program and what is going on
- Consequence:
  - A debugger does not help much if your program does not crash, e.g. just gives wrong results
  - Avoid using a debugger as far as possible.
  - Use it as last resort.

Programming Techniques

- Think about a verbose execution mode of your program
- Use a careful/paranoid programming style
  - check invariants and pre-requisites (assert(m>=0), assert(v<c))

Static Code Analysis – Compiler Flags

- Use the debugging/assertion techniques of the compiler
  - use debug flags (-g), warnings (-Wall)
  - array bound checks in Fortran
  - use memory debug libraries (-lefence)

skipped

Avoiding Debuggers

- Write portable programs
  - it avoids future problems
    - **architectures/platforms have a short life**
    - **all compilers and libraries have bugs**
    - **all languages and standards include implementation defined behavior**
  - running on different platforms and architectures significantly increases the reliability
- Use verification tools for parallel programming like assure

Valgrind – Debugging Tool

Rainer Keller

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
http://www.hlrs.de

Valgrind – Overview

- An Open-Source Debugging & Profiling tool.
- Works with any dynamically linked application.
- Emulates CPU, i.e. executes instructions on a synthetic x86.
- Currently it's only available for Linux/IA32.
- Prevents Error-swamping by suppression-files.
- Has been used on many **large** Projects:
  KDE, Emacs, Gnome, Mozilla, OpenOffice.

- It's easily configurable to ease debugging & profiling through *skins*:
  - **Memcheck**: Complete Checking (every memory access)
  - Addrcheck: 2xFaster (no uninitialized memory check).
  - Cachegrind: A memory & cache profiler
  - **Callgrind**: A Cache & Call-tree profiler.
  - Helgrind: Find Races in multithreaded programs.

Valgrind – Usage

- Programs should be compiled with
  - Debugging support (to get position of bug in code)
  - Possibly without Optimization (for accuracy of position & less false positives):
    `gcc –O0 –g –o test test.c`
- Run the application as normal as parameter to valgrind:
  `valgrind ./test`
- Then start the MPI-Application as with TV as debugger:
  `mpirun –dbg=valgrind ./mpi_test`

Valgrind – Memcheck

- Checks for:
  - Use of uninitialized memory
  - Malloc Errors:
    - **Usage of free'd memory**
    - **Double free**
    - **Reading/writing past malloced memory**
    - **Lost memory pointers**
    - **Mismatched malloc/new & free/delete**
  - Stack write errors
  - Overlapping arguments to system functions like `memcpy`.

Valgrind – Example 1/2

```
array = malloc (SIZE * sizeof (int));

if (rank == 0)
  MPI_Recv (array, SIZE+1, MPI_INT, 1, 4711, MPI_CO
else
  MPI_Send (array, SIZE+1, MPI_INT, 0, 4711, MPI_CO

printf ("(Rank:%d) array[0]:%d\n", rank, array[0]);
```


Valgrind – Example 2/2

- With Valgrind `mpirun –dbg=valgrind –np 2 ./mpi_murks`:

PID

```
==11278== Invalid read of size 1
==11278==   at 0x4002321E: memcpy (../../memcheck/mac_replace_strmem.c:256)
==11278==   by 0x80690F6: MPID_SHMEM_Eagerb_send_short (mpich/../shmemshort.c:70)
.. 2 lines of calls to MPIch-functions deleted ...
==11278==   by 0x80492BA: MPI_Send (/usr/src/mpich/src/pt2pt/send.c:91)
==11278==   by 0x8048F28: main (mpi_murks.c:44)
==11278== Address 0x4158B0EF is 3 bytes after a block of size 40 alloc'd
==11278==   at 0x4002BBCE: malloc (../../coregrind/vg_replace_malloc.c:160)
==11278==   by 0x8048EB0: main (mpi_murks.c:39)
```

Buffer-Overrun by 4 Bytes in MPI_Send

```
....

==11278== Conditional jump or move depends on uninitialised value(s)
==11278==    at 0x402985C4: _IO_vfprintf_internal (in /lib/libc-2.3.2.so)
==11278==    by 0x402A15BD: _IO_printf (in /lib/libc-2.3.2.so)
==11278==    by 0x8048F44: main (mpi_murks.c:46)
```

Printing of uninitialized variable

- It can not find:
  - May be run with 1 process: One pending Recv (Marmot).
  - May be run with >2 processes: Unmatched Sends (Marmot).

skipped

Valgrind – Calltree 1/2

- The Calltree skin (like the cachegrind skin):
  - Tracks memory accesses to check Cache-hit/misses.
  - Additionally records call-tree information.

```
==11745== Calltree-0.9.6, a cache profiler for x86-linux.
==11745== Copyright (C) 2002, and GNU GPL'd, by N.Nethercote and J.Weider
==11745== Using valgrind-2.1.0, a program supervision framework for x86-l
==11745== Copyright (C) 2000-2003, and GNU GPL'd, by Julian Seward.
--11745-- warning: Pentium with 12 K micro-op instruction trace cache
--11745--          Simulating a 16 KB cache with 32 B lines
==11745== Estimated CPU clock rate is 1410 MHz
==11745== For more details, rerun with: -v
--11745--
```

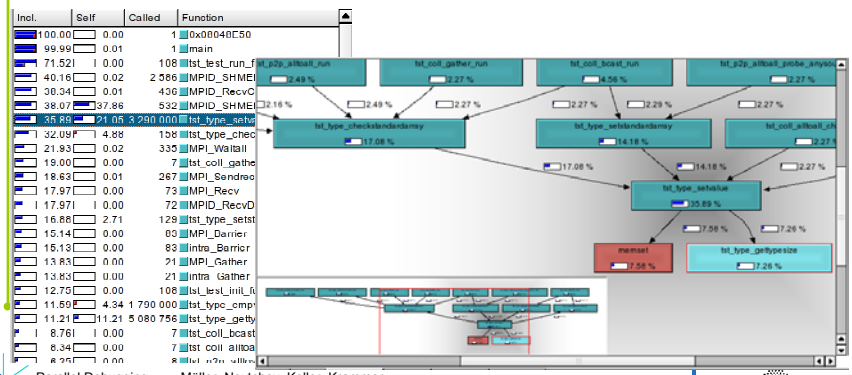
- After the run, it reports overall program statistics:

```
==11810== D   refs:      497.790.574  (386.176.612 rd + 111.613.962 wr)
==11810== D1  misses:        863.493  (    369.495 rd +     493.998 wr)
==11810== L2d misses:        282.232  (     98.857 rd +     183.375 wr)
==11810== D1  miss rate:         0.1% (        0.0%   +         0.4%  )
==11810== L2d miss rate:         0.0% (        0.0%   +         0.1%  )
```


skipped

Valgrind – Calltree 2/2

- Even more interesting: the output trace-file.
- With the help of kcachegrind, one may:
  - Investigate, where Instr/L1/L2-cache misses happened2-2.
  - Which functions were called where & how often.



Valgrind – Deficiencies

- Valgrind cannot find of these Error-Classes:
  - Semantic Errors
  - Timing-critical errors
  - Uninitialised stack-memory not detected.
  - Problems with new instruction sets
    (e.g. SSE/SSE2 is supported, certain Opcodes are **not**).
    When using the Intel-Compiler: `-tpp5` for Pentium optimisation.

MARMOT

Bettina Krammer

University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

What is MARMOT?

- MPI analysis and checking tool to verify at runtime if an application conforms to the MPI standard.
- Library written in C++ that will be linked to the application.
- No source code modification of the application is required.
- Additional process working as debug server, i.e. the application will have to be run with **`mpirun`** for n+1 instead of n processes.
- Implementation of C and Fortran language binding of MPI-1.2 standard.
- Environment variables for tool behaviour and output (report of errors, warnings and/or remarks, trace-back, etc.).
- After the execution of the program the user can read a logfile to check for potential problems.

Availability of MARMOT

- Tests on different platforms, using different compilers and MPI implementations, e.g.
  - IA32/IA64 clusters (Intel, g++ compiler) mpich
  - IBM Regatta
  - NEC SX5
  - Hitachi SR8000
- Download and further information
  http://www.hlrs.de/organization/tsc/projects/marmot/

Example 1: request-reuse (source code)

```
/*
** Here we re-use a request we didn't free before
*/

#include <stdio.h>
#include <assert.h>
#include "mpi.h"

int main( int argc, char **argv ) {
  int size   = -1;
  int rank   = -1;
  int value  = -1;
  int value2 = -1;
  MPI_Status  send_status, recv_status;
  MPI_Request send_request, recv_request;

  printf( "We call Irecv and Isend with non-freed requests.\n" );
  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &size );
  MPI_Comm_rank( MPI_COMM_WORLD, &rank );
  printf( " I am rank %d of %d PEs\n", rank, size );
```


Example 1: request-reuse (source code cont'd)

```
 if( rank == 0 ){
    /*** this is just to get the request used ***/
    MPI_Irecv( &value, 1, MPI_INT, 1, 18, MPI_COMM_WORLD, &recv_request );
    /*** going to receive the message and reuse a non-freed request ***/
    MPI_Irecv( &value, 1, MPI_INT, 1, 17, MPI_COMM_WORLD, &recv_request );
    MPI_Wait( &recv_request, &recv_status );
    assert( value = 19 );
  }
  if( rank == 1 ){
    value2 = 19;
    /*** this is just to use the request ***/
    MPI_Isend( &value, 1, MPI_INT, 0, 18, MPI_COMM_WORLD, &send_request );
    /*** going to send the message ***/
    MPI_Isend( &value2, 1, MPI_INT, 0, 17, MPI_COMM_WORLD, &send_request );
    MPI_Wait( &send_request, &send_status );
  }
  MPI_Finalize();
  return 0;
}
```

Example 1: request-reuse (output log)

```
   We call Irecv and Isend with non-freed requests.
1 rank 0 performs MPI_Init
2 rank 1 performs MPI_Init
3 rank 0 performs MPI_Comm_size
4 rank 1 performs MPI_Comm_size
5 rank 0 performs MPI_Comm_rank
6 rank 1 performs MPI_Comm_rank
   I am rank 0 of 2 PEs
7 rank 0 performs MPI_Irecv
   I am rank 1 of 2 PEs
8 rank 1 performs MPI_Isend
9 rank 0 performs MPI_Irecv
10 rank 1 performs MPI_Isend
   ERROR: MPI_Irecv Request is still in use !!
11 rank 0 performs MPI_Wait
   ERROR: MPI_Isend Request is still in use !!
12 rank 1 performs MPI_Wait
13 rank 0 performs MPI_Finalize
14 rank 1 performs MPI_Finalize
```


Parallel Debuggers

Parallel Debuggers

- Most vendor debuggers have some support
- gdb has basic support for threads
- Debugging MPI programs with a “scalar” debugger is hard but possible
  - MPIch supports debugging with gdb attached to one process
  - manual attaching to the processes is possible
- TotalView is a good but expensive tool
- DDT is an alternative

TOTALVIEW

What is TotalView?

- Parallel debugger
- Source level debugging for C, C++, F77, F90, HPF
- **MPI, OpenMP**, Pthreads, PVM, shmem
- SMPs, MPPs, PVPs, Clusters
- Available on all major Unix Platforms and most Supercomputers
- GUI (independent of platform, exception Cray T3E)
  - TotalView 4.x based on tcl/tk
  - TotalView 5.x based on Motif

Availability of TotalView

- Compaq Digital Alpha
  HP-UX
  IBM RS6000 and SP Power
  SGI MIPS
  Sun SPARC SunOS 5
  Linux Intel IA32 (RedHat)
  Linux Alpha (RedHat)
  Linux IA64
  Linux Opteron
- Cray T3E by Cray
- Hitachi SR2201 by SofTek, SR8000
- NEC SX Series

Availability of TotalView at HWW

Platform	Availability	Remarks
Volvox	Yes	V6
Hitachi SR8000	Yes	V 4.0
Cray T3E	Yes	Cray 3.0.0
NEC SX	Yes	
SGI Onyx	No	Use cvd

Development Platforms

More information:

http://www.hlrs.de/organization/tsc/services/tools/debugger/totalview

Availability of TotalView at University Stuttgart

- Two user 8 CPU Floating License for University Stuttgart:
  1. Download Software from http://www.etnus.com
  2. Set environment variable for license:

     LM_LICENSE_FILE=7244@servint1.rus.uni-stuttgart.de

More information about campus licenses available at

http://www.hlrs.de/organization/tsc/services/tools/campus

TotalView usage at HLRS

- Set USE_TOTALVIEW in your login scripts
- CRAY T3E: set USE_PROG_ENV
- Compile with -g compiler switch
  CRAY T3E: compiler switch -G
- command name: **totalview**

Starting TotalView

On a new process:

```
% totalview myprog -a arguments to myprog
```

To debug MPI programs:

```
% totalview mpirun -a -nprocs 3 myprog
% mpirun -tv -np 3 myprog
```

To debug IBM POE programs:

```
% totalview poe -a myprog [args]
```

To debug CRAY T3E programs:

```
% totalview -X #procs myprog [args]
```

skipped

TotalView on Hitachi SR8000

- Compilation:
  - `f90 -g`
  - `cc -g`
  - `KCC -g --backend -tv`
- OpenMP
  - `f90 -g -omp -procnum=8`
  - `cc -g -omp -parallel=1 -O2`
- MPI
  - `mpirun -tv`

skipped

TotalView on HPN

- Compilation:
  - `f90 -g`
  - `cc -g`
  - `KCC -g`
- OpenMP
  - `guidef90 -g`
  - `guidec -g`
  - `guidec++ -g`
- MPI
  - `mpirun -np #procs -tv ./a.out`

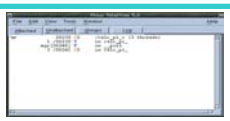
skipped

TotalView Exercise: Basic Look & Feel

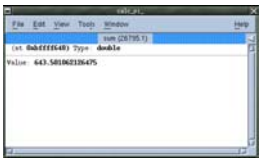
- Log into hwwhpn.hww.de
- Use bash as shell
- Change into directory
  `~/TOTALVIEW/#NR/TOTALVIEW/SIMPLE`
- Compile calc_pi_{f90,c,cc}.{f90,c,cc}
- Start totalview with `totalview` *`executable`*

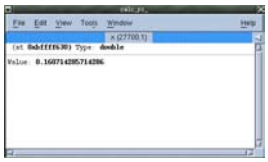
TotalView Windows

Root Window



Process Window

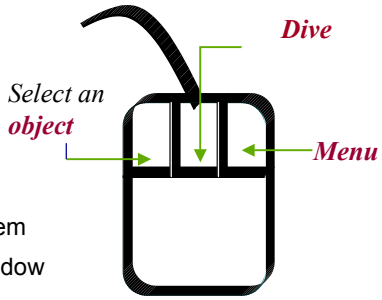




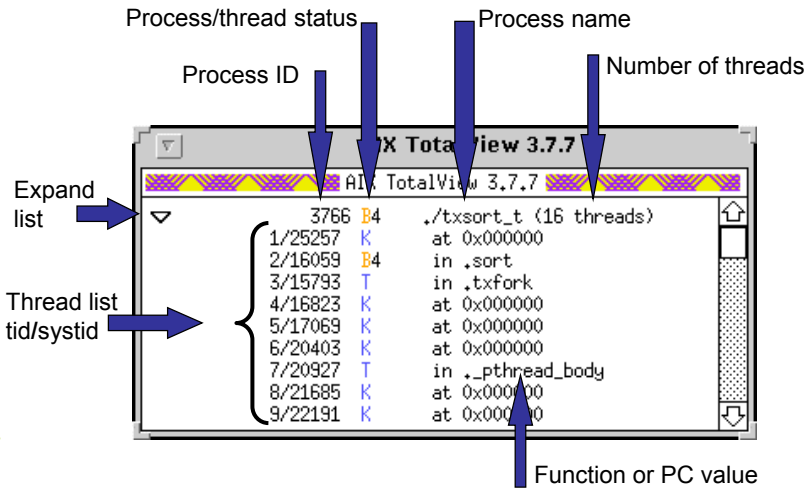
Data Windows

TotalView Mouse Buttons

- **Left** button is **Select**:
  - Chooses an item of interest,
  - Starts editing a item
- **Middle** button is **Dive**:
  - Gets more information about an item
  - **Shift+Dive** forces open a new window
- **Right** button is **Menu**:
  - Raises a menu of actions
  - All menus have a **Help** (^?) entry



TotalView Main Window



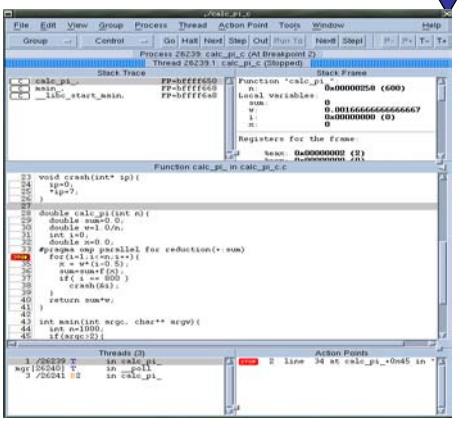
TotalView Process Window

Process/thread motion buttons

Stack Trace pane

Source pane

Thread pane



Local variables for the selected frame

Action Points pane

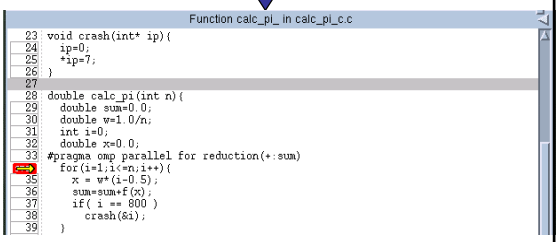
TotalView Source Pane

Gridded box is a possible site for a breakpoint

Current function and source file

Select to set one

Current point of execution



- **Dive** on a source word to get more information
- **Select** a line to use **Run to selection** command
- **Select** or **dive** on a line number to set an action point

Parallel Debugging - Philosophy

- By default, TotalView places processes in groups
  - Program Group - Includes parent and all related processes
  - Share Group - Only processes that share the same source code

- Command can act on single process or share group
  - halt process (h) , halt group (H)
  - next step process (n), next step group (N)
  - go process (g), go group (G)

skipped

TotalView Exercise: Debug simple program

- Run calc_pi inside totalview:
  - Check where the program crashes

- Analyze core file with totalview
  - run calc_pi
  - execute `totalview calc_pi core`

- For advanced users: choose another programming paradigm:
  - MPI, OpenMP, MPI+OpenMP

TotalView support for debugging MPI

- Special support for MPI is available depending on your MPI library:
  - display message queue state of a process
- Supported MPI implementations:
  - mpich v1.1.0 or later (use -debug in configure)
  - HP MPI v1.6
  - Compaq MPI >v1.7
  - IBM, release >2.2 of Parallel Environment, threaded version of MPI
  - SGI MPI v1.3 or later

TotalView MPI Message Queue Window

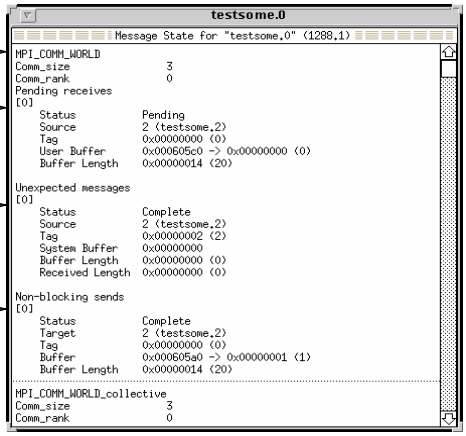
Communicator name and info

Non-blocking receive operations

Unmatched incoming messages

Non-blocking send operations

- Dive on source or target to refocus Process window
- Dive on buffer to see message contents



skipped

TotalView Exercise: Parallel program

- Example in TOTALVIEW/MPI:
  - deadlock_{c,cc,f90}.{c,cc,f90}
  - start program with `mpirun -tv -np 2 a.out`
  - interrupt execution after “deadlock”
  - try to find the reason for the deadlock and fix it
- For advanced users:
  - pending_{c,cc,f90}.{c,cc,f90}
  - try to find pending message by setting breakpoint at `MPI_Finalize`

TotalView more information

- http://www.etnus.com/products/totalview/index.html
- http://www.hlrs.de/organization/tsc/services/tools/debugger/totalview
  - User Guide
  - Installation Guide
  - CLI Guide
  - Powerpoint Tutorial
- CRAY T3E: Online Documentation at http://www.hlrs.de/platforms/crayt3e

Distributed Debugging Tool (DDT)

What is DDT?

- Parallel debugger
- Source level debugging for C, C++, F77, F90
- MPI, OpenMP
- SMPs, Clusters
- Available on Linux distributions and Unix
- GUI (independent of platform, based on QT libraries)

Availability of DDT

- Linux:
  - Linux IA32 (Intel and AMD)
  - Linux IA64
  - Linux Opteron
- Unix
  - PowerPC (AIX)
  - SGI Altix
  - SGI Irix
  - SUN Sparc
  - PA-Risc and Itanium Superdome

skipped

Availability of DDT at HWW

Platform	Availability	Remarks
Volvox	Yes	V 1.4
AzusA	Yes	V 1.4

More information:

http://www.hlrs.de/organization/tsc/services/tools/debugger/ddt/

Availability of DDT at University Stuttgart

- Two user Floating License for University Stuttgart:
  1. Download Software from http://www.etnus.com
  2. Set environment variable for license.

     LM_LICENSE_FILE=7244@servint1.rus.uni-stuttgart.de

More information about campus licenses available at

http://www.hlrs.de/organization/par/services/tools/campus

skipped

DDT usage at HLRS

- Set USE_DDT in your login scripts
- Compile with -g compiler switch
- Command name: **ddt or $DDT**
- To start debugging with DDT simply type:

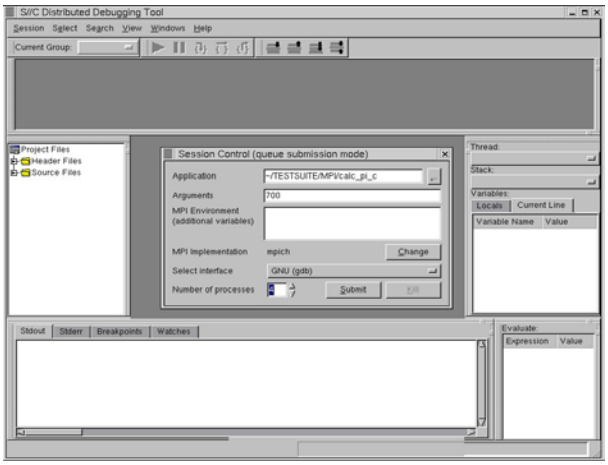
  % $DDT myprog *arguments to myprog*

DDT Look & Feel

DDT Main Window

Configuration Window

and all belonging Panes (Thread, Stack, Output, Source code, etc.)



DDT Main/Process Window

MPI Groups

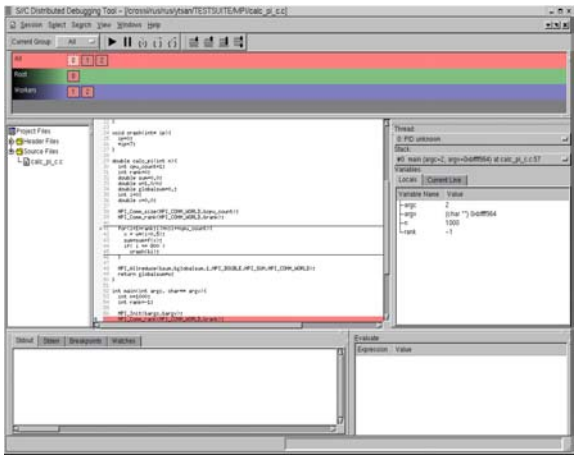
File browse and Source pane

Output,

Breakpoints,

Watch

Pane



Thread,

Stack,

Local and Global Variables

Pane

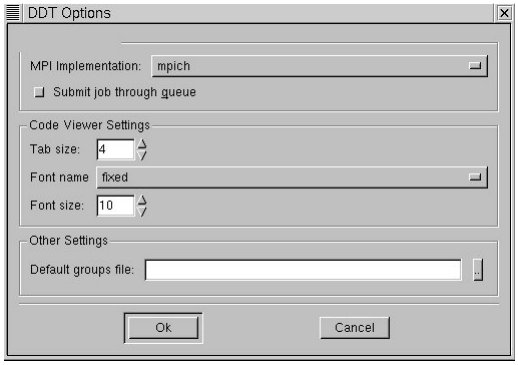
Evaluation window

skipped

DDT Options Window

The Options window: here is the MPI library implementation selected

The program can also be submitted through some batch system



skipped

DDT Options Window (Queue)

The Options windows: This program uses mpich as MPI implementation and starts the program through PBS batch system

For more information about the Template files and the commands, please read the DDT Users Manual.

skipped

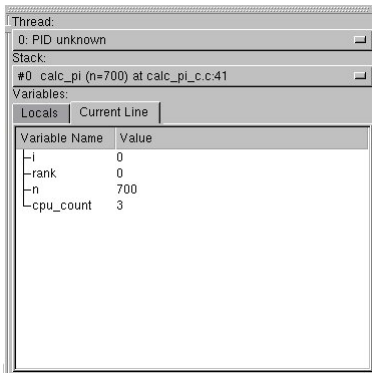
DDT Thread and Stack Window

Thread Pane:
Switch between all program threads

Stack Pane:
Switch between the functions in the selected thread

Variables Pane:
Local variables:
Shows the variables value for the function in which ddt is currently stopped

Current Line:
Shows every variable value between the two lines selected by the user



DDT Source Pane

With the right mouse button Set or Remove a breakpoint at the selected line

When the program is running and stopped at a breakpoint, the line is coloured in red (by OpenMP programs) and red, blue or green by programs using MPI

Parallel Debugging - Philosophy

- By default, DDT places processes in groups
  - All Group - Includes parent and all related processes
  - Root/Workers Group - Only processes that share the same source code
- Command can act on single process or group
  - stop process , stop group
  - next step process , next step group
  - go process, go group

DDT more information

- http://www.streamline-computing.com/softwaredivision_1.shtml