
Portable OpenMP debugging with TotalView

James Cownie

Etnus LLC.

jcownie@etnus.com

Shirley Browne

University of Tennessee

browne@cs.utk.edu

Design Goals

Allow users to see all the state of their OpenMP programs

Expose threaded nature of OpenMP execution model

Allow users to

- debug threaded code
- see how execution reached the current point
- access private and shared variables

Make OpenMP debugging no worse than sequential debugging



TotalView Support for OpenMP

Source level debugging of original OpenMP code

Ability to plant breakpoints throughout OpenMP code

Visibility of OpenMP worker threads

Access to SHARED and PRIVATE variables in OMP
PARALLEL code

Access to OMP THREADPRIVATE data on some platforms



Platforms and Compilers with TotalView OpenMP Support

MIPSpro 7.3 or later on SGI IRIX

Compaq Fortran on Tru64 UNIX Alpha

IBM xlf on AIX

**KAI Guide (C and Fortran) on all the platforms we both
support.**



OpenMP Compiler Code Transformations

Outlining

Calls to OpenMP runtime library

Splitting of variables between the original routine and outlined routine

- Shared variables in master thread's original routine
- Private variables in outlined routine

Creation of worker threads to share work load in parallel region

Threadprivate common blocks and static variables



Impact of Transformations

A single OpenMP source line appears in two functions

Code executing in a thread is called by the runtime

- User can't see the "real" caller
- Stack backtraces are useless

A single PRIVATE OpenMP source variable is replicated in each thread.

Shared variables are accessed via pointers or the static link

Accessing thread static data is complicated



.....

Thread Debugging

TotalView was already a thread-capable debugger before
OpenMP support was added

User level thread libraries already supported

User cannot single-step into OpenMP region

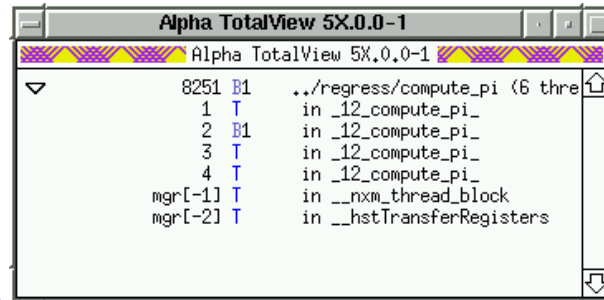
- Next line actually executed in a different thread
- Instead place breakpoint inside parallel region

Control over multithreaded execution within parallel region
depends on OS support for asynchronous thread control.

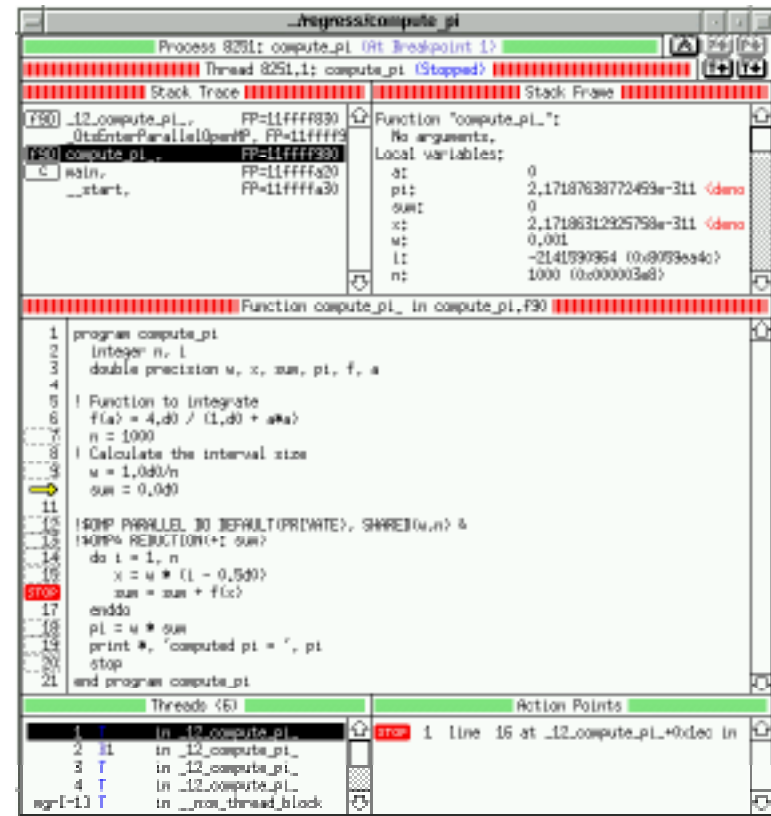


Sample OpenMP Debugging Session

Totalview
root
window
can show
all threads.



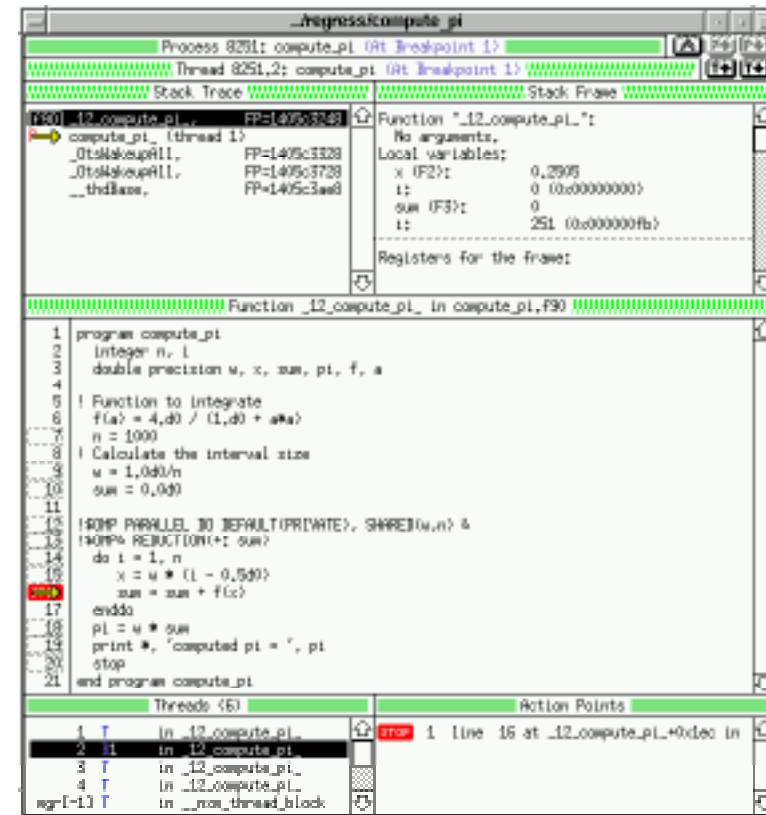
Process window
shows one thread.
In this case the
master



Stack Backtraces

TotalView inserts a special stack parent token line in the stack trace pane of OMP worker threads when they are stopped in an outlined routine.

Selecting the parent link refocuses the display onto the parent thread and stack frame, thus showing how a specific execution point was reached.

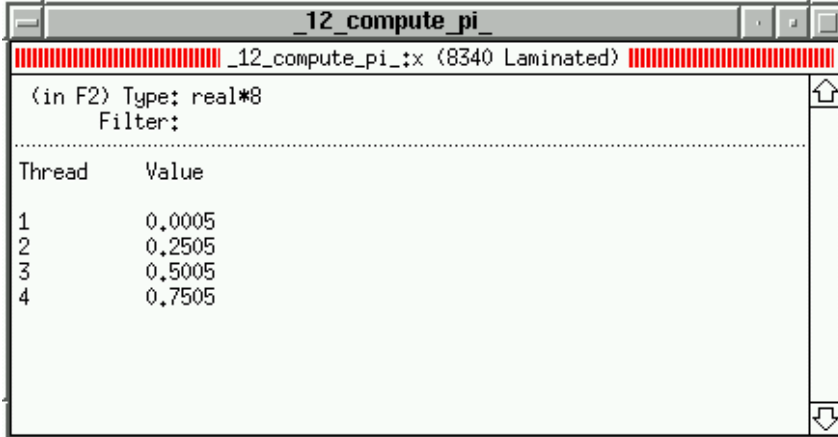


Private and Shared Variables

Private variables are stored by the compiler like local variables.

Shared variables are maintained in the master thread's original routine stack frame.

Thread Laminated display can display values of a private variable in all threads that have a matching stack frame.



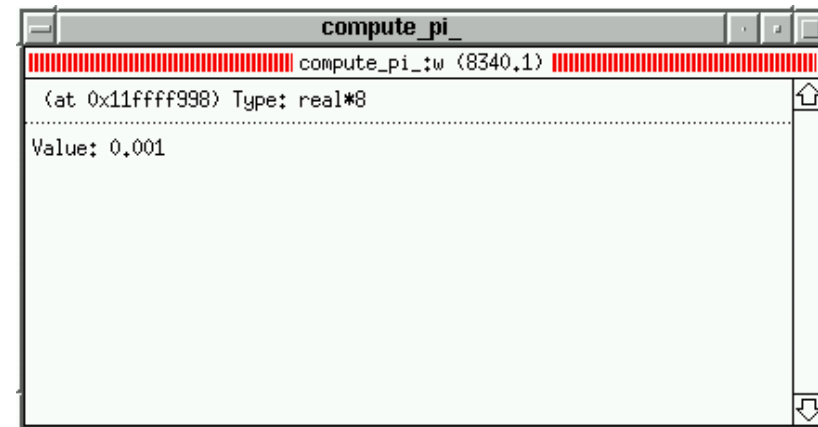
Thread	Value
1	0,0005
2	0,2505
3	0,5005
4	0,7505



OMP Shared Variable

Diving on the shared variable `W` raises a data window showing the value from the parent function.

- Note the colour coding of the window to distinguish each thread's data.
- The same window will be raised whichever thread the dive is performed from.



OMP THREADPRIVATE Common Blocks

Different compilers and runtime systems use different (complicated) implementation techniques.

TotalView support on

- Compaq Tru64 UNIX
- SGI IRIX
- AIX (with latest compilers)



Debugging Hybrid MPI/OpenMP

Already supported – addition of OpenMP debugging capability is orthogonal to existing capabilities as a multiprocess MPI enabled debugger.

OpenMP threads can be displayed for each MPI process.

Main issue is the availability of thread safe MPI implementations.



Future Work

Locks

Barrier state

OMP variable information

**Asynchronous control (will be available in TotalView 5.0
later this year).**

Additional compilers



Conclusions

To debug OpenMP codes you need to understand how the compiler is transforming your code

TotalView lets you see what is going on at runtime

Debugging multi-threaded codes is never going to be as simple as debugging sequential codes

We're doing all we can to make it easy

Hiding the parallelism is the wrong thing to do

