Debugging Tools

Multi-threaded Programming, Tuning and Optimization on Multi-core MPP Platforms
February 15-17, 2011
CSCS Manno
Potential Errors in Multithreaded Codes (1)

- Data Race Condition
  - Two or more threads accessing and updating same memory location
  - Unintended and unsynchronized access to shared variable resulting in unexpected behavior

\[
\begin{align*}
\text{Time} & \\
\text{Thread 1 reads (x)} & \quad \text{Thread 1 modifies (x)} & \quad \text{Thread 1 writes (x)} \\
\text{Thread 2 reads (x)} & \quad \text{Thread 1 updates and writes (x)} \\
\text{Thread 3 reads (x)} & \\
\end{align*}
\]
Potential Errors in Multithreaded Codes (2)

- **Deadlocks**
  - Threads waiting on a resource that never becomes available

```
Time

Thread A

Lock Resource # 1

Thread A

Wait for Resource # 2

Thread B

Lock Resource # 2

Thread B

Wait for Resource # 1
```
Deadlock Examples

!$OMP PARALLEL DEFAULT(SHARED)

!$OMP CRITICAL
   DO I = 1, 10
      X = X + 1
   !$OMP BARRIER
   Y = Y + I*I
   END DO
!$OMP END CRITICAL
!$OMP END PARALLEL

#pragma omp parallel
   myid = omp_get_thread_num();
   if (myid %2) {
      // do some odd work
      #pragma omp barrier
      // do more work
   }
   else {
      // do some even work
      #pragma omp barrier
      // do more work
   }
}
How To Avoid Deadlocks

• Using Barrier in for selected number of threads

• Avoid the lock functions as much as possible

• Avoid nesting of locks
Potential Errors in Multithreaded Codes (3)

• Livelock
  – Infinite loop conditions for certain loops (traditional loop errors)

• Memory issues
  – Stack overflows due to replication of private data items
  – Memory leaks
Debugging

What is going on inside your application?

A debugger will follow the program through execution so you can watch your program execute step by step and view the contents of memory:

* Let you examine the program execution step by step
* Make the program stops on specified places or on specified conditions
* Give information about current variables’ values, the memory and stack

[Re]-run (No recompilation Is needed)

Insert Breakpoints

Analyse
Important Debugging Concepts

• Stopping and watching a program execution at a certain point
  – Breakpoints
  – Watchpoints

• Stepping and continuing to control program execution
  – Single step lines of code
  – Single step assembler instructions
  – Resume program execution

• Examining the stack
  – Backtracing
Debugging Considerations Unique to OpenMP

• Challenges
  – Nondeterministic failures
  – Reproducibility and predictability of bugs
  – Compiler transformation manifests complex bugs
  – Memory analysis for data scoping constructs

• Use multi-threading feature of a debugger to:
  – Identify exactly when the failure occur (thread level)
  – Review program execution and memory contents
  – Inspect core files after a crash
Debugging Tools Available on CSCS Platforms

• Two tools available at the moment
  – Totalview (for hybrid MPI+OpenMP applications)
    • module avail xt-totalview
  – gdb
    • module avail gdb

• Must compile with the -g compiler flag

• For failures or crashes that occur after long execution times
  – Cray Fast Track Debugging (compile with -gFast)—only available for Cray compilers
  – For further info: http://docs.cray.com/books/S-9401-0909//S-9401-0909.pdf
Other Tools to Aid Multi-threaded Programming

• Debugging tools
  – Allinea DDT (MPI + OpenMP)
  – Debugger from different compiler vendors
  – …

• Correctness tools
  – Intel Inspector
  – Rogue Wave ThreadSpotter
  – …
Further Info

- Debugging and Performance Analysis Tools available at CSCS [http://www.cscs.ch/145.0.html](http://www.cscs.ch/145.0.html)
- Craydocs [http://docs.cray.com/](http://docs.cray.com/)
- Totalview tools (debugger, memory scope and replay engine) [http://www.totalviewtech.com/home/](http://www.totalviewtech.com/home/)
- Acumem ThreadSpotter and SlowSpotter [http://www.acumem.com](http://www.acumem.com)
TotalView is a debugger with support for Fortran, C, C++, MPI, OpenMP and threads.

TotalView is an interactive tool that lets you debug serial, multiprocessor and multithreaded programs. It can be executed either as a graphical user interface (by using the totalview executable) or from a command-line interface (by using the totalviewcli executable). Totalview provides source-level debugging of Fortran and Fortran 90, C, and C++ codes. It can be used to debug parallel programs based on MPI. It also has facilities for multi-process thread-based parallel programs such as OpenMP.
Breakpoints

Execution Toolbars

Set Breakpoint

Co-operating Processes

Multi-threading programming

02/2011
### Root window

#### Multi-threading programming

**Host name**

- **Action Point ID number**
- **Process Status**
- **Expand - Collapse Toggle**
- **TotalView Thread ID #**
- **Rank # (if MPI program)**
- **Hierarchical/Linear Toggle**

<table>
<thead>
<tr>
<th>ID</th>
<th>Rank</th>
<th>Host</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>&lt;local&gt;</td>
<td>B</td>
<td>mismatchLinux.0 (1 active threads)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux (5)</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.1 (5)</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.2 (5)</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.3 (5)</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.4 (5)</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.5 (5)</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.6 (5)</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>intrepid.etnus.c</td>
<td>T</td>
<td>/home/barryk/tests/fork_loopLinux.7 (5)</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>&lt;local&gt;</td>
<td>B</td>
<td>mismatchLinux.1 (1 active threads)</td>
</tr>
<tr>
<td>13.1</td>
<td>1</td>
<td>&lt;local&gt;</td>
<td>B4</td>
<td>in main</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>&lt;local&gt;</td>
<td>B</td>
<td>mismatchLinux.2 (1 active threads)</td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>&lt;local&gt;</td>
<td>B</td>
<td>mismatchLinux.3 (1 active threads)</td>
</tr>
</tbody>
</table>
Multi-threading programming
02/2011

Action points

Set Evaluation Point

Enter Stopping Condition
Examining data

Dive in And Visualize

Multi-threading programming
02/2011
Viewing data across processes

Dive into myid
Memory usage

View Memory Usage At Breakpoint

Multi-threading programming
02/2011
Memory leaks

View Memory Leaks At Breakpoint

Multi-threading programming
02/2011
Demo : Totalview
Multi-threading programming

Getting started (1)

```bash
/bin/bash 60x38

program affinity

(!$
  use omp_lib

#ifdef _MPI
  use mpi
#endif

IMPLICIT NONE

integer, external :: running_on
character(len=8) :: nid
integer :: coreid
integer :: rc=0, rank=0, numtask=0, corespercn=24
integer :: threadid=0, nthreads=0

#ifdef _MPI
  call MPI_INIT ( rc )
call MPI_COMM_RANK ( MPI_COMM_WORLD, rank, rc )
call MPI_COMM_SIZE ( MPI_COMM_WORLD, numtask, rc )
#endif

 !$omp parallel private(threadid,coreid,nid)
$!
threadid = omp_get_thread_num()
$!	nthreads = omp_get_num_threads()

call print_nodeaffinity(nid)
coreid = running_on()
print *,
"rank=" , rank , ",/" , numtask,
" cnid=" , trim(nid(1:8)) , 
" threadid=" , threadid , ",/" , nthreads , 
" core=" , coreid

$!omp end parallel

#endif

call MPI_FINALIZE ( rc )
#endif

end program affinity

"aff3.F90" 37 lines --2%--}
```
Getting started (2)

```bash
palu1: ls
aff2.F90 Makefile runningon.c
palu1: make FFLAGS="-g -fopenmp -D_MPI"
cc -g -fopenmp -D_MPI -c runningon.c
ftn -g -fopenmp -D_MPI -c aff2.F90
ftn -g -fopenmp -D_MPI -o exe runningon.o aff2.o
/usr/lib/../lib64/libpthread.a(sem_open.o): In function `sem_open':
/usr/src/packages/BUILD/glibc-2.9/nptl/sem_open.c:330: warning: the use of `mktemp' is dangerous, better use `mkstemp'
palu1:
palu4:
palu1: salloc -N2
salloc: Granted job allocation 2959
palu4: squeue -u $USER

+---------------------------------+---------------------------------+-----------------+-----------------+----------------+----------------+----------------+
| JOBSID | USER      | ACCOUNT | NAME     | PARTITION | ST       | EXEC_HOST | REASON | START_TIME | TIME | TIME_LEFT | NODES | PRIORITY |
+--------+-----------+---------+----------+-----------+----------+----------+--------+------------+------+-----------+-------+----------+
| 2959   | piccinal  | csstaff | bash     | day       | R        | palu1    | None   | 2011-02-09T09:58 | 0:08 | 59:52     | 2     | 2        |
+--------+-----------+---------+----------+-----------+----------+----------+--------+------------+------+-----------+-------+----------+

palu4:
palu4: export OMP_NUM_THREADS=2 ; aprun -n2 -N1 -d24 exe
rank= 0 / 2 cnid=nid00003 threadid= 1 / 2 core= 1
rank= 0 / 2 cnid=nid00003 threadid= 0 / 2 core= 0
rank= 1 / 2 cnid=nid00004 threadid= 0 / 2 core= 0
rank= 1 / 2 cnid=nid00004 threadid= 1 / 2 core= 1
done
done
Application 108245 resources: utime ~0s, stime ~0s
palu1: exit
exit
salloc: Relinquishing job allocation 2959
salloc: Job allocation 2959 has been revoked.
palu4:  
```
Launching Totalview (1)

**File**
- ID
- Rank
- Host
- Status
- Description

---
1
<local>
- 
aprun <0 active threads>

**Edit**

**View**

**Tools**

**Window**

**Help**

**Group (Control)**

- Do
- Halt
- Kill
- Restart
- Next
- Step
- Out
- To
- Go Back
- Pray
- Unt
- Step
- Call
- Lane

**Debugger**

- Stack Trace
- Stack Frame

**Function main in aprun.c**

Parallel program has not yet been started.

**Multi-threading programming**

02/2011
Launching Totalview (2)

Multi-threading programming
02/2011
Inserting Breakpoints

Multi-threading programming
02/2011
Viewing data across processes and threads
Restarting and exiting Totalview

```
Application: 138251 resource: utime "Os, stime "Os
The TotalView Debugger Server has died
psall:  export OMP_NUM_THREADS=2 ; totalview aprun -a -N1 -d24 exe
Linux x86_64 TotalView 8.8.0-1
Copyright 2007-2010 by TotalView Technologies, LLC. ALL RIGHTS RESERVED.
Copyright 1998-2007 by Etnus, LLC.
Copyright 1998 by Etnus, Inc.
Copyright 1996-1998 by Dolphin Interconnect Solutions, Inc.
Copyright 1988-1996 by BBN Inc.
TotalView Technologies ReplayEngine
Copyright 2010 TotalView Technologies
ReplayEngine uses the UndoDB Reverse Execution Engine
Copyright 2005-2010 Undo Limited
Reading symbols for process 1, executing "aprun"
Library /usr/bin/aprunc, with 2 assets, was linked at 0x00000000 and initially loaded at 0xffffffff00000000
Mapping 256131 bytes of ELF string data from '/usr/bin/aprunc'...done
Indexing 25600 bytes of DWARF '.debug_frame' symbols from '/usr/bin/aprunc'...done
Indexing 25144 bytes of DWARF '.debug_info' symbols from '/usr/bin/aprunc'...done
Library @syscall_library-64, with 1 assets, was linked at 0xffffffff00000000, and initially loaded at 0x77f7b0000000: Using previously cached local copy of library '@syscall_library-64'
Mapping 82 bytes of ELF string data from '@syscall_library-64'...done
Indexing 82 bytes of DWARF '.eh_frame' symbols from '@syscall_library-64'...done
Library /lib64/libdl.so.2, with 2 assets, was linked at 0x00000000 and initially loaded at 0xffffffff00000000
Mapping 459 bytes of ELF string data from '/lib64/libdl.so.2'...done
Indexing 420 bytes of DWARF '.eh_frame' symbols from '/lib64/libdl.so.2'...done
Library /lib64/libthread.so.0, with 2 assets, was linked at 0x00000000, and initially loaded at 0xffffffff00000000
Mapping 14508 bytes of ELF string data from '/lib64/libthread.so.0'...done
Indexing 12504 bytes of DWARF '.eh_frame' symbols from '/lib64/libthread.so.0'...done
Library /lib64/libc.so.6, with 2 assets, was linked at 0x00000000 and initially loaded at 0xffffffff00000000
Mapping 21676 bytes of ELF string data from '/lib64/libc.so.6'...done
Indexing 12500 bytes of DWARF '.eh_frame' symbols from '/lib64/libc.so.6'...done
Library /lib64/libX11/libX11.so.6, with 2 assets, was linked at 0x00000000, and initially loaded at 0xffffffff00000000
Mapping 380 bytes of ELF string data from '/lib64/libX11/libX11.so.6'...done
Indexing 8072 bytes of DWARF '.eh_frame' symbols from '/lib64/libX11/libX11.so.6'...done
Library /lib64/libnss_files.so.2, with 2 assets, was linked at 0x00000000, and initially loaded at 0xffffffff00000000
Mapping 7007 bytes of ELF string data from '/lib64/libnss_files.so.2'...done
Indexing 3250 bytes of DWARF '.eh_frame' symbols from '/lib64/libnss_files.so.2'...done
Launching TotalView Debugger Servers with commands:
  /opt/toolworks/totalview/8.8.0a/Linux-x86-64/bin/tvcsrvmain -verbose info 172.26.0.31
INFO: Using previously cached local copy of library /usr/local/lib/108253.exe
Mapping 32443 bytes of ELF string data from '/usr/local/lib/108253.exe'...done
Indexing 73727 bytes of DWARF '.debug_frame' symbols from '/usr/local/lib/108253.exe'...done
Indexing 115132 bytes of DWARF '.debug_info' symbols from '/usr/local/lib/108253.exe'...done
Mapping 1048576 bytes of DWARF '.debug_info' symbols from '/usr/local/lib/108253.exe'...done
Reading symbols for process 2, executing "./exe"
Reading symbols for process 3, executing "./exe"
rank= 0 / 2 cnid=100000 threadid= 0 / 2 core= 0
rank= 0 / 2 cnid=100001 threadid= 0 / 2 core= 1
rank= 1 / 2 cnid=100002 threadid= 0 / 2 core= 1
rank= 1 / 2 cnid=100003 threadid= 0 / 2 core= 1
done
Application: 138251 resource: utime "Os, stime "Os
psall: exit
exit
salloc: Reclaiming Job allocation 2960
salloc: Job allocation 2960 has been revoked.
psall: "}
Multi-threading programming
Frequently used GDB commands

General Commands
- **help [name]**: Show information about GDB command
- **run [<args>]**: runs selected program with arguments <args>
- **attach <pid>**: attach gdb to a running process
- **Kill**: kills the process being debugged
- **Quit**: quits the gdb program

Stepping and Continuing
- **continue**: continue execution (after a stop)
- **step**: step one line, entering called functions
- **next**: step one line, without entering functions

Breakpoint commands
- **break [<where>]**: sets breakpoints. <where> can be a function name, a line number or a hex address
- **watch <expr>**: sets a watchpoint, which will break when <expr> is written to [or read]
- **info breakpoints**: prints out a listing of all breakpoints
- **delete [<nums>]**: deletes breakpoints

Commands for looking around
- **list [<where>]**: prints out source code at <where>
- **backtrace [<n>]**: prints a backtrace <n> levels deep
- **info [<what>]**: prints out info on <what>
- **print [<expr>]**: prints out <expr>
- **display**: prints value of expression each time the program stops
Demo: GDB
Getting started

Multi-threading programming
02/2011
Launching GDB
Inserting Breakpoints

```
palu2: export OMP_NUM_THREADS=3 ; aprun -n 1 -d24 gdb .omp
Could not find platform independent libraries <prefix>
Could not find platform dependent libraries <exec_prefix>
Consider setting SPYTHONHOME to <prefix>[:<exec_prefix>]
'import site' failed; use -v for traceback
Traceback (most recent call last):
  File "<string>", line 32, in ?
ImportError: No module named os.path
GNU gdb (GDB) 7.2
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>
Reading symbols from /users/piccinal/AFF/palu/GNU/omp...done.
(gdb) break 34
Breakpoint 1 at 0x4004c6: file aff2.F90, line 34.
(gdb) break 45
Breakpoint 2 at 0x4003de: file aff2.F90, line 45.
(gdb) info break
    Num  Type  Disp  Enb  Address  What
  1  breakpoint  keep  y  0x0000000004004c6 in MAIN .omp fn.0 at aff2.F90:34
  2  breakpoint  keep  y  0x0000000004003de in affinity at aff2.F90:45
(gdb)
```
Viewing data across threads (1)

```
/gdb help run
Start debugged program. You may specify arguments to give it.
Args may include "*", or "[...]"; they are expanded using "sh".
Input and output redirection with ">", "<", or "" are also allowed.

With no arguments, uses arguments last specified (with "run" or "set args").
To cancel previous arguments and run with no arguments,
use "set args" without arguments.
/gdb run
Starting program: /users/piccin/ir/AFF/palu/GNU/omp
[Thread debugging using libthread_db enabled]
[New Thread 0x2aaaac9c950 (LWP 13736)]
[New Thread 0x2aaaaaaad950 (LWP 13737)]

Breakpoint 1, MAIN_omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
34   coreid = running_on()
/gdb info thread
   3 Thread 0x2aaaaaaad950 (LWP 13737) MAIN_omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
   2 Thread 0x2aaaac9c950 (LWP 13736) MAIN_omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
   1 Thread 0x6d38c0 (LWP 13733) MAIN_omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
/gdb print threadid
$1 = 0
/gdb list
   coreid = running_on()
   ! barrier
   print *, &
      "rank=",rank,"/",numtask, &
      "cnid=",%trim(nid(1:8)), &
      "threadid=",%trim(threadid,"/",nthreads, &
      "core=",%trim(coreid
   !$omp end parallel
```

Multi-threading programming
02/2011
Viewing data across threads (2)

```
/bin/bash 96x31

(gdb) info thread
  3 Thread 0x2aaaaaaaa950 (LWP 13737) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
  2 Thread 0x2aaaaaaaaac950 (LWP 13736) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
* 1 Thread 0x6d38c0 (LWP 13733) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
(gdb) thread 2

(Switching to thread 2 (Thread 0x2aaaaaaaaac950 (LWP 13736)))#0  MAIN__omp_fn.0 (.omp_data_i=0x0)
  at aff2.F90:34
34  coreid = running_on()
(gdb) info thread
  3 Thread 0x2aaaaaaaa950 (LWP 13737) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
  2 Thread 0x2aaaaaaaaac950 (LWP 13736) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
  1 Thread 0x6d38c0 (LWP 13733) MAIN__omp_fn.0 (.omp_data_i=0x0) at aff2.F90:34
(gdb) print threadid
$s = 1
(gdb) thread 3

(Switching to thread 3 (Thread 0x2aaaaaaaa950 (LWP 13737)))#0  MAIN__omp_fn.0 (.omp_data_i=0x0)
  at aff2.F90:34
34  coreid = running_on()
(gdb) print threadid
$s = 2
(gdb) help cont
Continue program being debugged, after signal or breakpoint.
If proceeding from breakpoint, a number N may be used as an argument,
which means to set the ignore count of that breakpoint to N - 1 (so that
the breakpoint won't break until the Nth time it is reached).

If non-stop mode is enabled, continue only the current thread,
otherwise all the threads in the program are continued. To
continue all stopped threads in non-stop mode, use the -a option.
Specifying -a and an ignore count simultaneously is an error.
(gdb)
```

```
/bin/bash 50x36

24 !$  threadid = omp_get_thread_num()
25 !$  nthreads = omp_get_num_threads()
26 !!! corespercn = omp_get_num_procs()
27 !!!  MAXT = OMP_GET_MAX_THREADS()
28 !!!  INPAR = OMP_IN_PARALLEL()
29 !!!  DYNAMIC = OMP_GET_DYNAMIC()
30 !!!  NESTED = OMP_GET_NESTED()
31
32  call print_nodeaffinity(nid)
33  ! call print_coreaffinity(coreid)
34  ! coreid = running_on()
35  ! barrier
36  !
37  ! "rank", "rank", "/", ntasks,
38  ! "cnid", "trim(nid(1:))",
39  ! "threadid", "threadid ", "/", nthreads,
40  ! "core", coreid
41 !omp end parallel
42 #ifdef MPI
43  call MPI_FINALIZE ( rc )
44 #endif
45  print *, "done"
46  end program affinity
```
Restarting and exiting GDB

Restarting and exiting GDB

```
//bin/bash 96x31
at aff2.F90:34
34    coreid = running_on()
(gdb) print threadid
$3 = 2
(gdb) help cont
Continue program being debugged, after signal or breakpoint.
If proceeding from breakpoint, a number N may be used as an argument,
which means to set the ignore count of that breakpoint to N - 1 (so that
the breakpoint won't break until the Nth time it is reached).

If non-stop mode is enabled, continue only the current thread,
otherwise all the threads in the program are continued. To
continue all stopped threads in non-stop mode, use the -a option.
Specifying -a and an ignore count simultaneously is an error.
(gdb) cont
Continuing.
[Switching to Thread 0x2aaaaaaead950 (LWP 13737)]

Breakpoint 1, MAIN__omp_fn.0 (.omp.data_i=0x0) at aff2.F90:34
34    coreid = running_on()
(gdb) quit
A debugging session is active.

Inferior 1 [process 13733] will be killed.

Quit anyway? (y or n) [answered Y; input not from terminal]
Application 108323 resources; utime ~1s, stime ~0s
palu2: exit
salloc: Relinquishing job allocation 2982
salloc: Job allocation 2982 has been revoked.
palu2:
```
• Questions?
• Hands-on exercises
• Thank you for your attention
Data race condition

- A data race occurs under the following conditions:
  - 2 or more threads in a process concurrently access the same memory location,
  - At least one of the threads is accessing the memory location for writing, and
  - The threads are not using any exclusive locks to control their accesses to that memory.
- When these three conditions hold, the order of accesses is non-deterministic. Therefore each run can give different results depending on the order of the accesses. Some data races may be harmless (for example, when the memory access is used for a busy-wait), but many data races are either bugs or caused by bugs in the program.

The object of this exercise is to determine whether it's safe to parallelise every DO loop that you see. Follow these steps:
- copy loopy.f90 to your directory.
- compile the code sequentially (that is with no '-fopenmp' flag) and determine the correct result.
- parallelise every loop and run the program on 2, 6, 12 and 24 threads (you can do this interactively) and compare these results with those from above.
- What's wrong?
- Rewrite your parallelised code to give the correct results irrespective of the number of threads used.
Segmentation faults

- Default thread stack size can be easy to exhaust. OpenMP thread stack size is an implementation dependent resource. In this case, the array is too large to fit into the thread stack space and causes the segmentation fault.

- The OpenMP standard does not specify how much stack space a thread should have. Consequently, implementations will differ in the default thread stack size.

- Default thread stack size can also be non-portable between compilers. Threads that exceed their stack allocation may or may not seg fault. An application may continue to run while data is being corrupted.

- **OMP_STACKSIZE** (OMP/3.0) : controls the size of the stack for (non-master) threads.

- Set the default thread stack size (in kilobytes by default) or B, K, M or G (bytes, kilobytes, megabytes or gigabytes).

The object of this exercise is to use the debuggers to find the origin of the segmentation fault. Follow these steps:

- copy `crash.f` to your directory.
- `module load PrgEnv-gnu gdb`
- compile (with `-g -fopenmp` flag) and run the code with any number of threads.
- What's wrong?
If OMP_STACKSIZE is not set, the initial value of the stacksize-var internal control variable is set to the default value.

MemoryScape only shows information for the main thread’s stack.
Deadlock

- Deadlock describes a condition where two or more threads are blocked (hang) forever, waiting for each other. Suppose we have a process with two or more threads. A deadlock occurs when the following three conditions hold:
  - Threads already holding locks request new locks,
  - The requests are made concurrently, and
  - Two or more threads form a circular chain where each thread waits for a lock that the next thread in the chain holds.

- Here is an example of a deadlock condition:
  - Thread 1: holds lock A, requests lock B
  - Thread 2: holds lock B, requests lock A

- A deadlock can be of two types: A "potential deadlock" or an "actual deadlock". A potential deadlock is a deadlock that did not occur in a given run, but can occur in different runs of the program depending on the timings of the requests for locks by the threads. An actual deadlock is one that actually occurred in a given run of the program. An actual deadlock causes the threads involved to hang, but may or may not cause the whole process to hang.