HPC Programming

Debugging, Part II

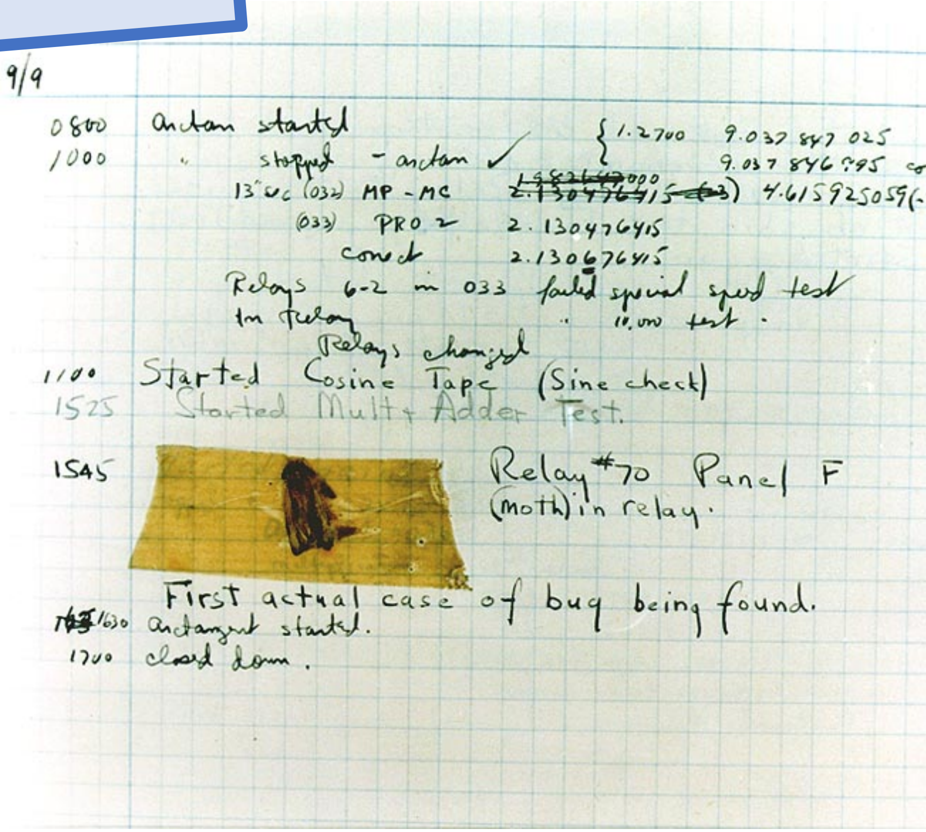
Peter-Bernd Otte, 22.1.2019

Definition of a bug

Recap

- “bug” := errors or glitches in a program
  → incorrect result.

- most difficult part of debugging: finding the bug.
  Once found, correcting is relatively easy
  - prove: bug bounty programs
  - debuggers: help programmers locate bugs by: executing code line by line, watching variable values
- locating bugs is something of an art:
  - why? a bug in one section of a program cause failures in a completely different section
  - there is no defined right way to debug

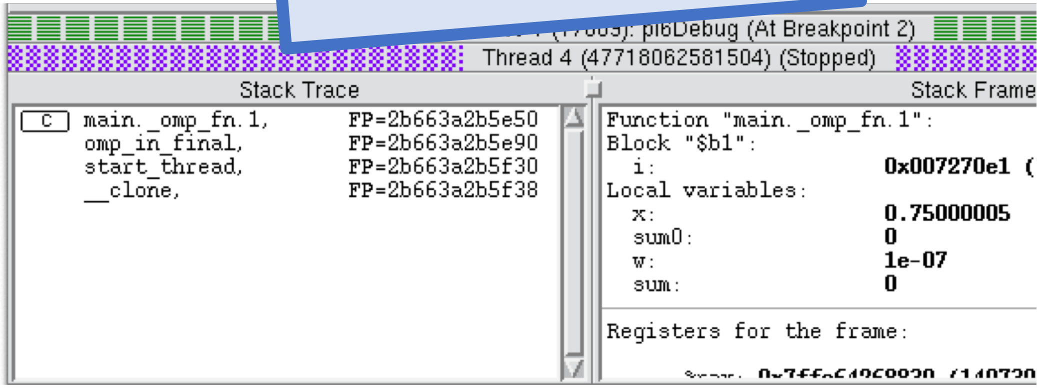


1946, moth removed from relay

Call Stack and Program Counter

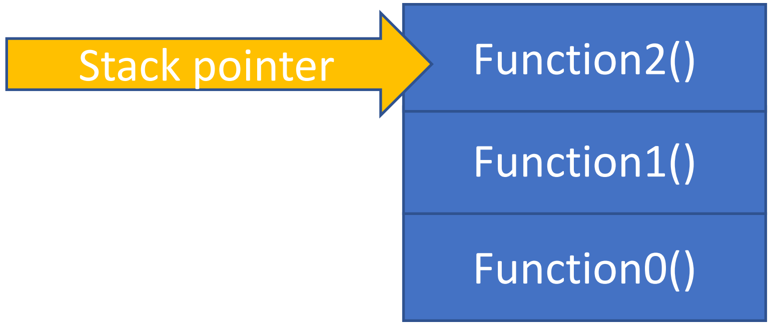
Recap

- Call stack = stack of "stack frames"
  - LIFO (last in, first out)
  - Function call → new stack frame.
    Removed when call ends



- Program Counter (PC):
  - Hardware register in processor, indicating the actual point in program sequence.
  - Stack Frame includes a return address
    → PC can be reset at end of called subfunction

- Stack pointer:
  - Address register, that points to the top of the call stack



Hints

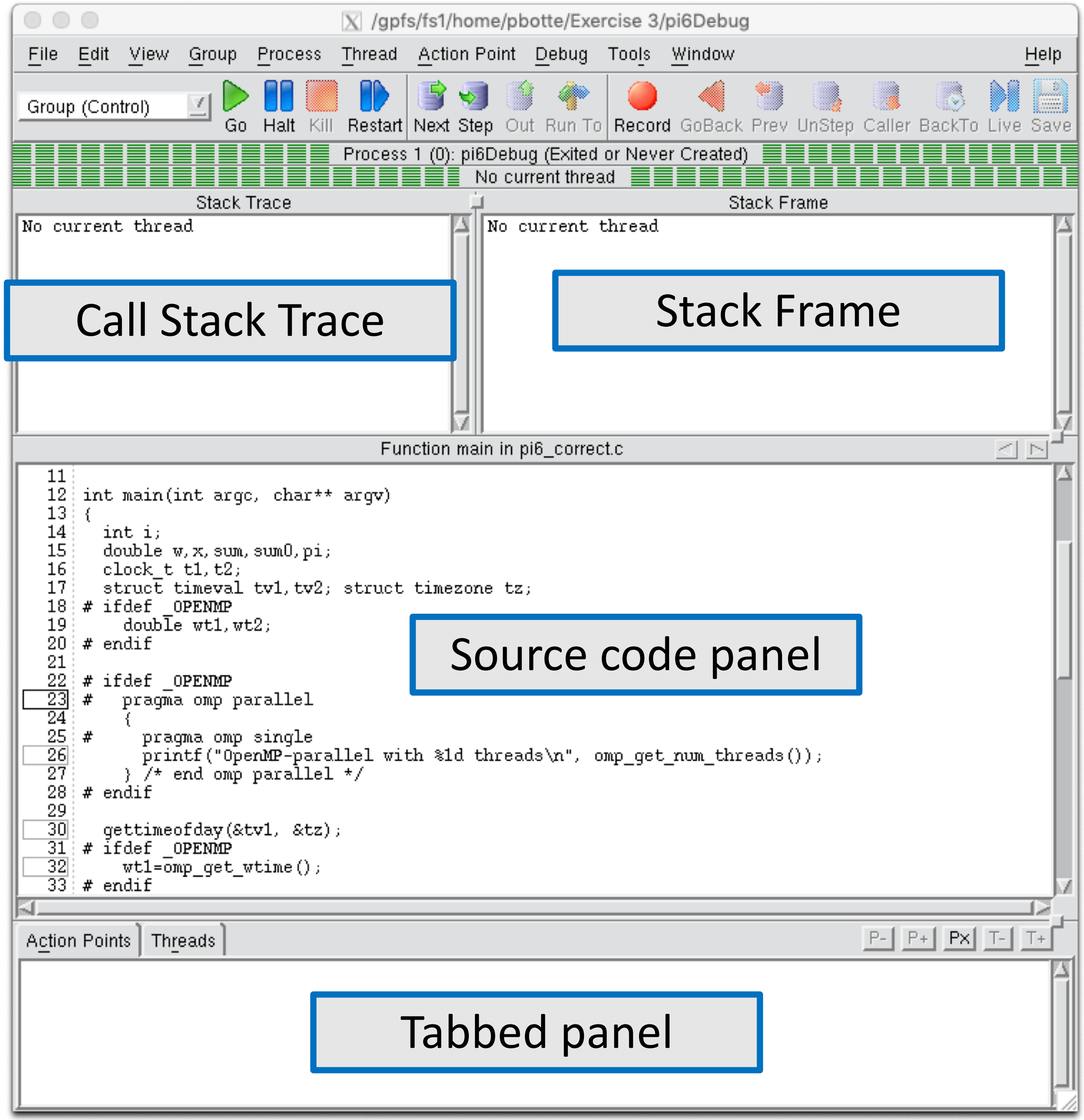
Recap

- Think before coding --> Software Engineering
- Problem?
  1. remove all object, intermediate or temporary files
  2. Rebuild with debugging info on (-g) and optimisation off (-O0)
  3. Still problematic? --> debugger!

- Debug first a serial version of your program
- Some errors only occur
  - With optimized code (possible reasons: initialized variables? Wrong pointers? Buffer overflow?)
  - Outside of debug session (possible reason: different timing?)
  - With many processes

TotalView

- "Standard tool" for parallel debugging (OpenMP, MPI, CUDA)
- Wide compiler (Python, C, Fortran) and platform support (Linux, Unix, MacOS, no Windows)
- Process window:
  - State of one process / thread
- Last lecture: Stepping, Diving, Breakpoints, Watchpoints



Post-Mortem Analysis

Recap

Status: enabled on nodes, login not yet

Process does segmentation fault etc.

1. In bash: “ulimit -c unlimited” (check with ulimit –a and look for “core file size”)
2. Build your app with -O0 and –g and run
3. Test: ”kill -s SEGV <PID>”
4. Core file will be generated in same directory
5. Analyse with
   “totalview executable coreFileName”
   (or “gdb executable coreFileName”)

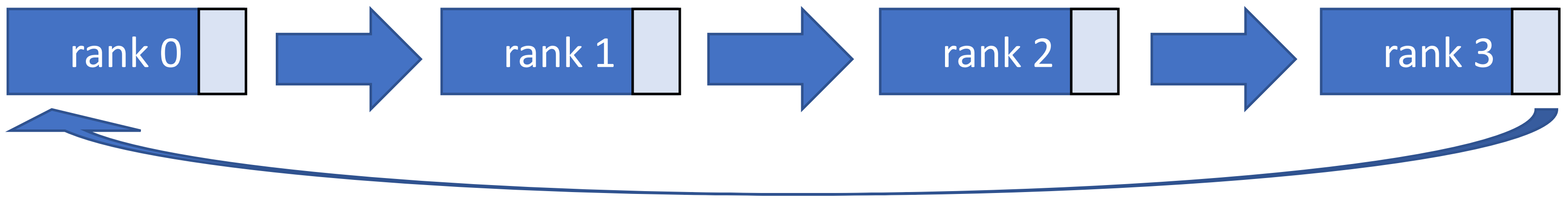
- Currently not allowed on Himster2, only backtrace (this will change)
- Hint: With “gcore <pid> -o <filename>“ a core dump is being generated and program remains running.

Debugging

1. Introduction / General Debugging
2. Typical bugs
3. Tools Overview
4. Introduction TotalView
5. Debugging with TotalView OpenMP
6. Debugging with TotalView MPI

Deadlocks, Race condition

- see lectures from OpenMP and MPI
- Deadlock: cyclic list, all threads proceed when receive OK from predecessor



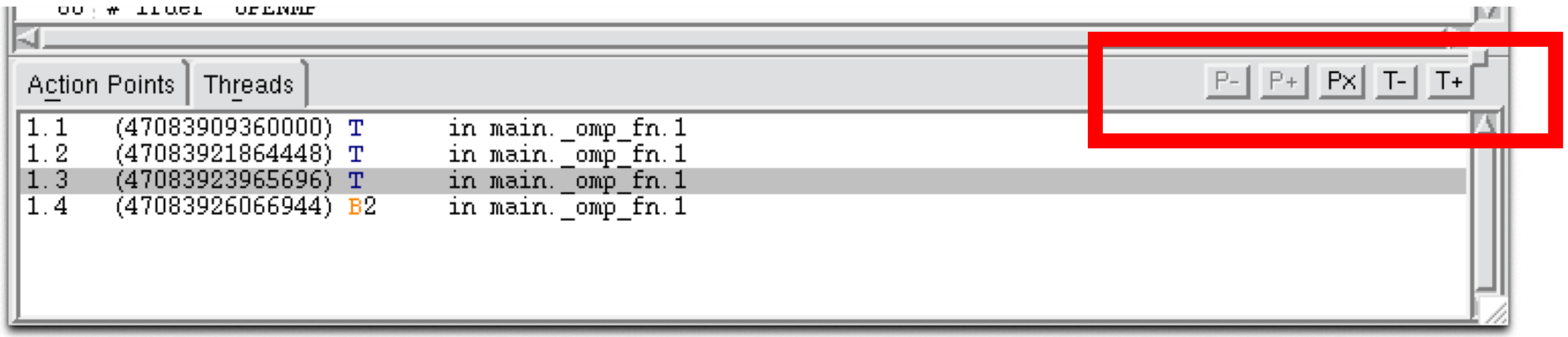
→ see exercise 4 today.

- Race Condition: multiple threads, shared resources, result depends on scheduler

We will go directly to the hands-on part

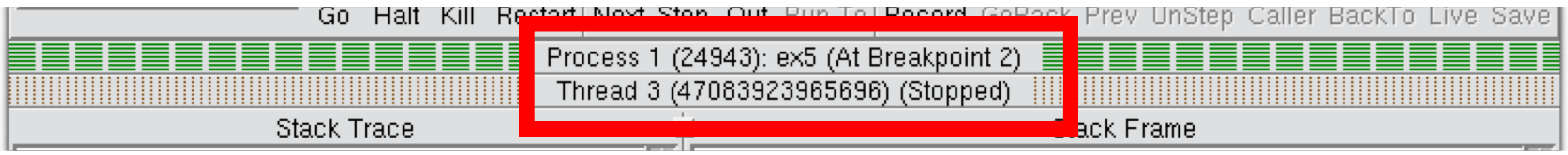
Parallel Debugging Hints

- During runtime, change between Threads and Processes:



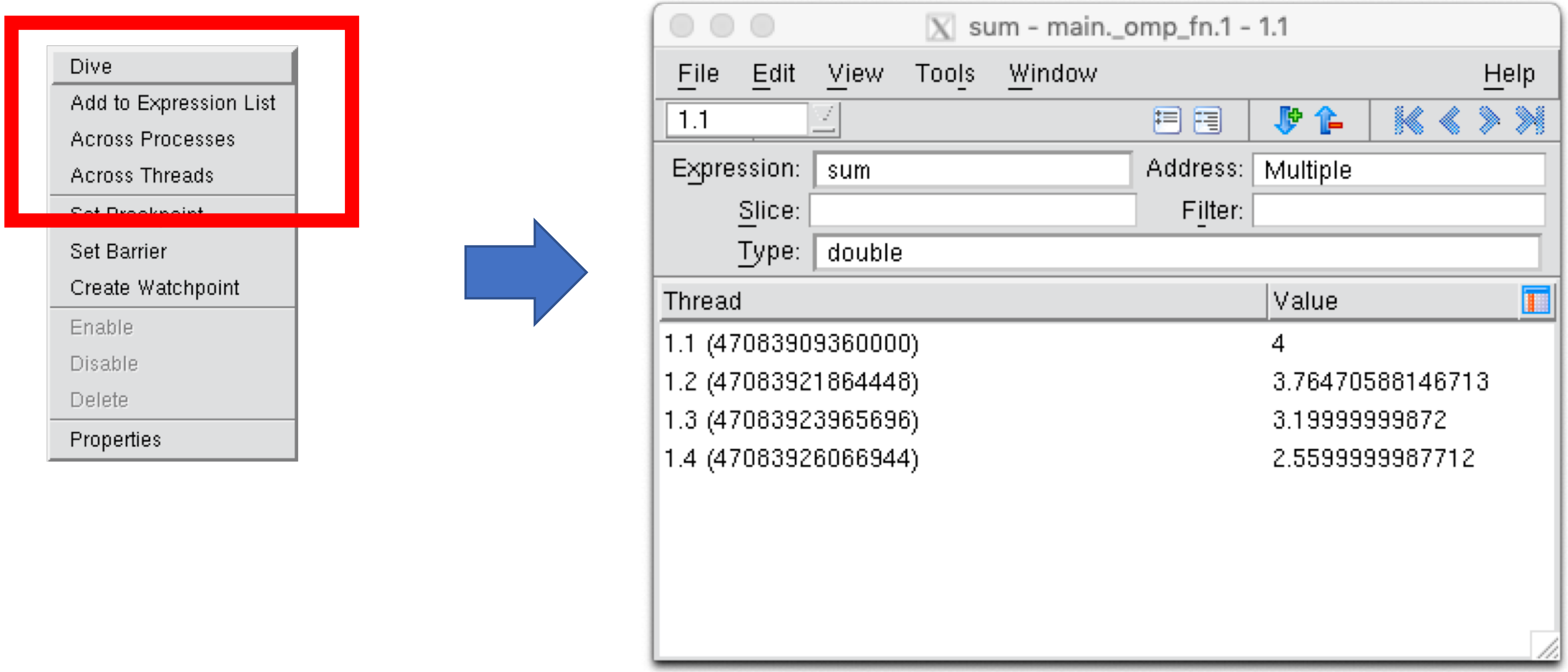
Shows the current status of threads and processes

- The headline will show the current process / thread:



Parallel Debugging Hints

- Dive into Variables across Threads and Processes (right click)



Set up your workbench

- Connect two times via SSH to Mogon2 / HIMster2 and work on the head node

  1. Use the first SSH connection for editing (gedit, vi, vim, nano, geany) and compiling
     $ compiling: gcc -g -O0 -o ExecutableName SourceFileName.c

  2. Use the second connection for the interactive usage of TotalView:
     $ module load debugger/TotalView/2018.0.5_linux_x86-64
     $ totalview &

- For MPI:
  - module load mpi/OpenMPI/3.1.1-GCC-7.3.0
  - Compile with: mpicc -g -O0 -o ExecutableName SourceFileName.c
  - Run on head node with: mpirun -n 2 ./ExecutableName

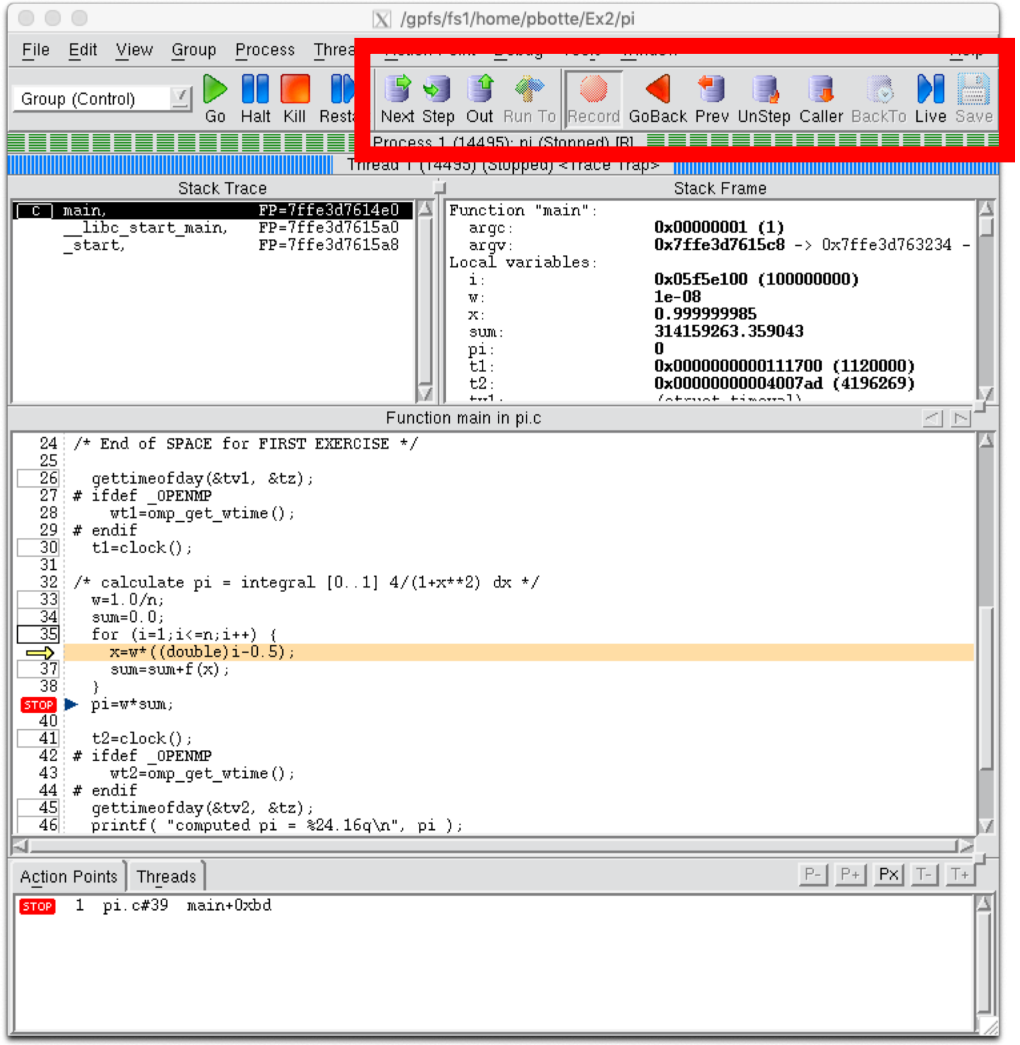
Exercise 2:

Learning objectives:

- TotalView: Replay Feature
- Note: this process slows down everything by order of magnitudes!

Steps:

1. Download the skeleton from OpenMP exercise 2 from lecture webpage:
   - wget [https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/02.zip && unzip 02.zip](https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/02.zip)
2. Compile as a single core not optimised (-g -O0) program and run this program with totalview. Activate the ReplayEngine when setting up in the debug options.
   1. Set a breakpoint at ”pi=w*sum;” run the program.
   2. dive into the variable sum and
   3. Go back in time with “Prev” (and forth with “next”) and check the value of sum



Exercise 3:

Learning objectives:

- TotalView: OpenMP

Steps:

1. Download the solution from OpenMP lecture 3, exercise 5 from lecture webpage:
   - wget https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/OpenMP-ex5_solution.zip && unzip ...
2. Compile as multithreaded:
   cc -fopenmp -O0 -g -o ExecutableName SourceFileName.c
   and run this program with totalview on many cores (OMP_NUM_THREADS=4).
3. Change the number of threads (in the menu under: Process > Startup Parameters) and run again.
4. Check the result with 1, 2, 4 and 8 threads in the team. Why is it different.
5. Find out the reason for this, by stopping the program before the sum gets reduced. Dive into the variable sum across threads (by right clicking it during runtime).
   HINT: If you do not manage to stop TotalView before the reduction takes place, use solution from exercise 4:
   https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/OpenMP_ex4_solution.zip

Exercise 4:

Learning objectives:

- TotalView: MPI

Steps:

1. Download the solution from MPI lecture 6, exercise 4 from lecture webpage:
    - wget https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/files/MPI-04-solution.zip && unzip ...
    - Load the corresponding MPI module before launching totalview
2. Launch TotalView, set up a “File > New Debug Parallel Program…” and select “Open MPI” as the Parallel System. Select 2 Tasks (or more) in the “Parallel Settings”, hit “next” and choose your executable. Run!
3. Compile and run with 2 processes. Stop the program after some MPI-data has been exchanged between the ranks (eg break point at line 54).
   Dive into variables across processes.
4. Bonus: Change the code to get a blocking situation, see MPI lecture 6, exercise 4, step 3: https://www.hi-mainz.de/fileadmin/user_upload/IT/lectures/WiSe2018/HPC/Lecture_HPC_6.pdf

   Debug this situation to familiarise with TotalView and parallel debugging.