Programming of Super Computers

Assignment 2: Parallel Debugging with TotalView

Isaías A. Comprés Ureña
compresu@in.tum.de

Prof. Michael Gerndt
gerndt@in.tum.de

17-11-2017

1 Introduction

Ensuring correctness is an important part of the software development process. In addition to correctness, the time required to produce results can also determine the usefulness of the software. Scientific software is often parallelized to produce results in less time. In the specific case of scientific software, the timeliness of the solution may be of equal importance as its correctness.

Unfortunately, while the parallelization process improves performance and allows an application to produce timely results, it also negatively affects the ability of developers to ensure correctness. In this second assignment, we will focus on a tool that helps us ensure the correctness of scientific software: the parallel debugger.

Stable free software parallel debuggers with support for distributed memory systems are not yet available. Because of this, in this assignment a commercial parallel debugger will be used: the TotalView debugger. TotalView is offered by Rogue Wave Software. The Leibniz Compute Centre (LRZ) provides access to it (via a module) to all users of the SuperMUC. This debugger allows for the inspection of source code directly, control of threads and processes, memory and state inspection, etc. It supports C, C++ and Fortran applications as well as CUDA for NVIDIA GPU kernels in recent versions.

In this assignment, students will learn how to prepare and build a parallel application so that it can be analyzed with this debugger. The students will first have to understand common issues that arise when developing OpenMP and MPI applications. Afterwards, they will learn how to find, identify and fix errors in the application with the aid of the parallel debugger.

2 Submission Instructions and General Information

Your assignment submission for Programming of Supercomputers will consist of 1 part:

- A 5 to 10 page report with the required answers.

Please refer to Section 2 of Assignment 1 for general instructions and recommended tools.

3 Understanding Parallel Programming Challenges

Parallel programming is necessary to take advantage of supercomputing hardware. Due to the wide availability of multi-core CPUs, in recent years parallel programming is now also necessary to take advantage of most desktop and laptop PC hardware, and even mobile devices.

Going from single-threaded to parallel programming can be challenging. It may require a complete redesign of the software: replacement of algorithms and data-structures, addition of locks, careful management

of memory and file descriptors, creation and tracking of threads and processes, etc. During these redesign activities, many common errors may be introduced into the software. It is important to understand these common errors so that they can be avoided, and in the worst case identified, located and fixed.

3.1 Task 1 (40 points total)

In this task, investigate and describe briefly in the report the following concepts:

1. Race Condition (1 point)
2. Deadlock (1 point)
3. Heisenbug (observer's effect) (1 point)
4. Cache Coherency and False Sharing (1 point)
5. Load Imbalance (1 point)
6. Amdahl's law (1 point)
7. Strong scaling applications (1 point)
8. Weak scaling applications (1 point)
9. Parallelization Overhead (1 point)
10. Floating-point arithmetic challenges (4 points)
    (a) Comparisons
    (b) Definition of a zero and signed zeros
    (c) Cancellation or loss of significance
    (d) Amplification and error propagation

Using figures here to illustrate the concepts is recommended.

3.1.1 Questions

1. Which of the concepts affect performance but not correctness? (2 points)
2. Which of the concepts affect the correctness of the application? (2 points)
    (a) Of these, which are exclusive to parallel programming? (2 points)
    (b) Of these, which are not exclusive to parallel programming?(2 points)
3. Which of them can occur in OpenMP applications? (2 points)
4. Which of them can occur in MPI applications? (2 points)
5. Is cache coherency necessary on MPI applications with a single process and a single thread per rank? Explain. (6 points)
6. Is Amdahl's law applicable to strong scaling applications? Explain. (3 points)
7. Is Amdahl's law applicable to weak scaling applications? Explain. (3 points)
8. Which of these limit the scalability of applications? (3 points)

4 Introduction to TotalView

TotalView is a parallel debugger offered by Rogue Wave Software. It is supported in several large compute centers in the world. With TotalView, developers can inspect source code, control threads and processes, inspect memory and state, etc. It supports C, C++ and Fortran applications, as well as multiple threads with OpenMP and multiple processes with MPI.

Before continuing with the subsequent tasks, take some time to learn the basics of TotalView. There are several online resources directly from Rogue Wave Software, as well as from some compute centers and universities. Here are some of them:

- Official TotalView Documentation from Rogue Wave Software
- TotalView at Leibniz Supercomputing Centre
- TotalView tutorial at Laurence Livermore National Labs

The Leibniz Compute Centre (LRZ) provides access to it (via a module) to all users of the SuperMUC. Some of the instructions in the LRZ's website (in the above list) are outdated, but the instructions on how to load and launch TotalView are correct.

The TotalView installation in SuperMUC seems to only work correctly in the "Phase 1" nodes, at the moment. Please make sure that you login to the "Phase 1" nodes and that you enable X11 forwarding and compression for your session:

```
ssh -YC <lrz-user>@sb.supermuc.lrz.de
```

You will need to use the IBM MPI and TotalView modules for this assignment. Make sure that you load them in your session:

```
module load totalview
module load mpi.ibm
```

The IBM MPI module is loaded by default, so it may not have to be loaded again. Make sure that you have the correct modules loaded by issuing the following command:

```
module list
```

and make sure that its output matches the following:

```
Currently Loaded Modulefiles:
 1) admin/1.0   4) intel/16.0 7) lrz/default
 2) tempdir/1.0 5) mkl/11.3   8) totalview/8.14
 3) lrztools/1.0 6) poe/1.4   9) mpi.ibm/1.4
```

Once you have made sure that you logged in to the "Phase 1" login nodes of the SuperMUC, and that you have the correct modules to work with TotalView, you can now proceed to build your benchmark.

4.1 Preparing the Benchmark for TotalView

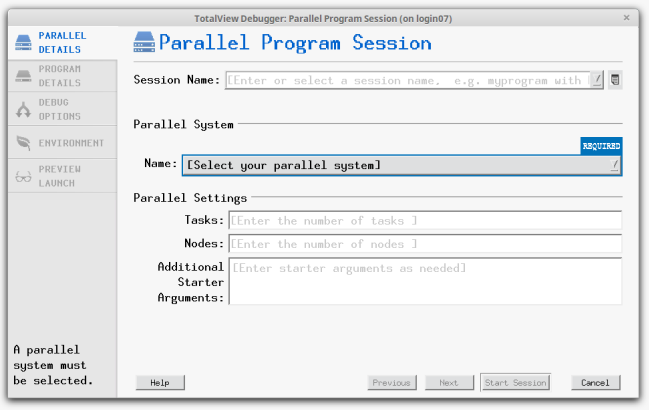
Refer to the instructions in assignment 1 and build 2 of the types of binaries: MPI and MPI+OpenMP. This can be done by building with the correct makefile options and then renaming the created binaries to avoid confusion. Additionally, make sure that the compiler flags are set to:

```
-g -O2
```

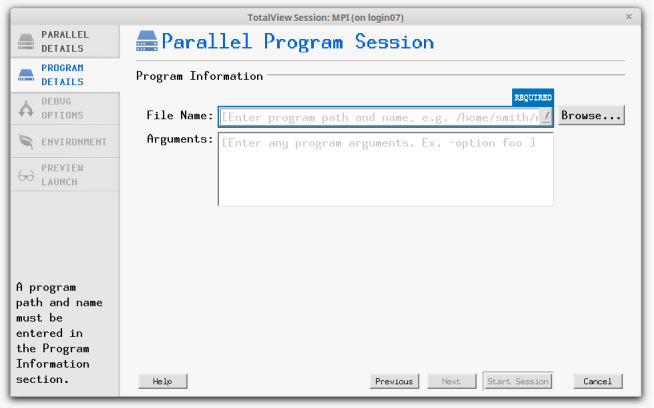
This is specified in the LRZ's TotalView instructions page. Please note that higher levels of optimizations will affect the ability of TotalView to match your source code, so please enable only the flags specified above. Rebuild your benchmark so that the new flags are now in effect.

4.1.1 Session Creation

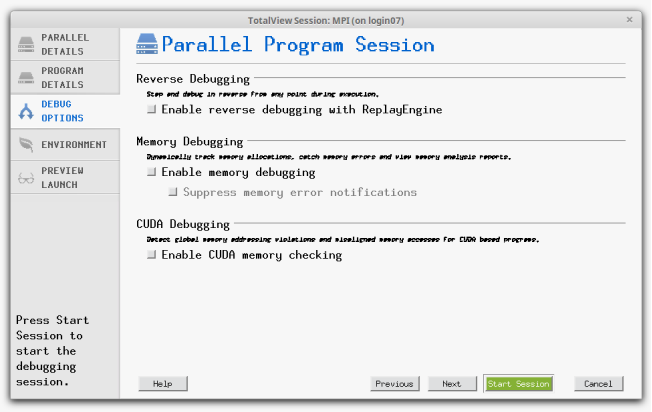
To set up a session, you can go back to the session manager by using the menu `File->Manage Sessions...` from the main TotalView window, or using the welcome screen once you restart TotalView. Select `A new parallel program`:



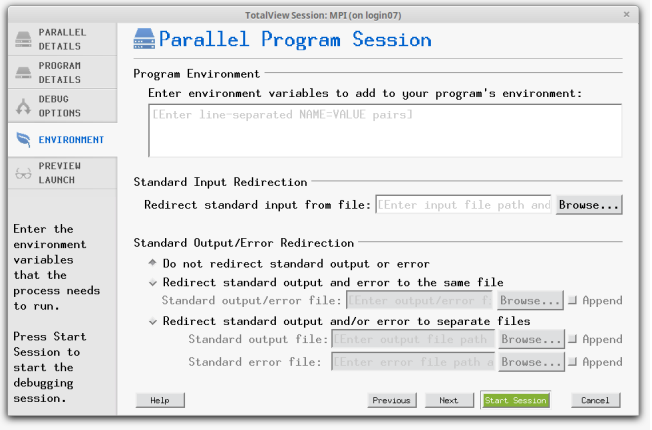
Specify a name for the new session. In the SuperMUC, the `Parallel System` needs to be selected in the drop-down menu. Pick `poe - Linux`. Under `Parallel Settings`, set the `Tasks (-procs)` option to the required number of processes to run the benchmark in 2 nodes. Each of the thin nodes has 16 cores, so make sure that you check in your benchmark's documentation (refer to Assignment 1) that you pick the maximum number that can be run in 2 nodes. Set the `Nodes (-nodes)` field to 2.



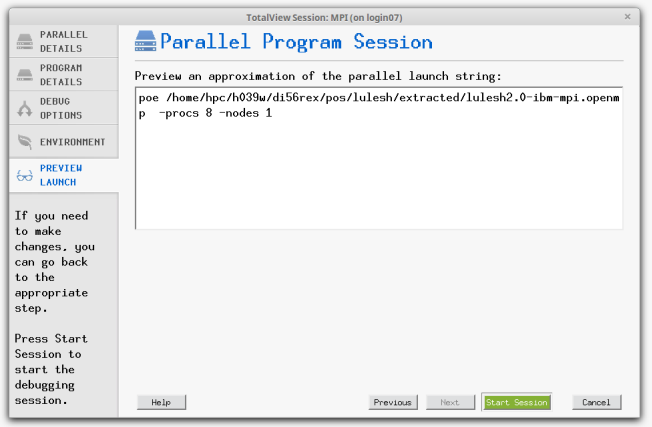
In the `PROGRAM DETAILS` tab, make sure to pick the correct binary for the session by clicking in the `Browse...` button and using the file dialog. You need to fill the `Arguments` field with the parameters for your application, if any.



The `DEBUG OPTIONS` screen comes next. You can enable `Memory Debugging` later if needed, at the cost of performance. Make sure that you do not enable `CUDA Debugging`, since the SuperMUC nodes do not have NVIDIA GPUs. Click the `Next` button to continue:

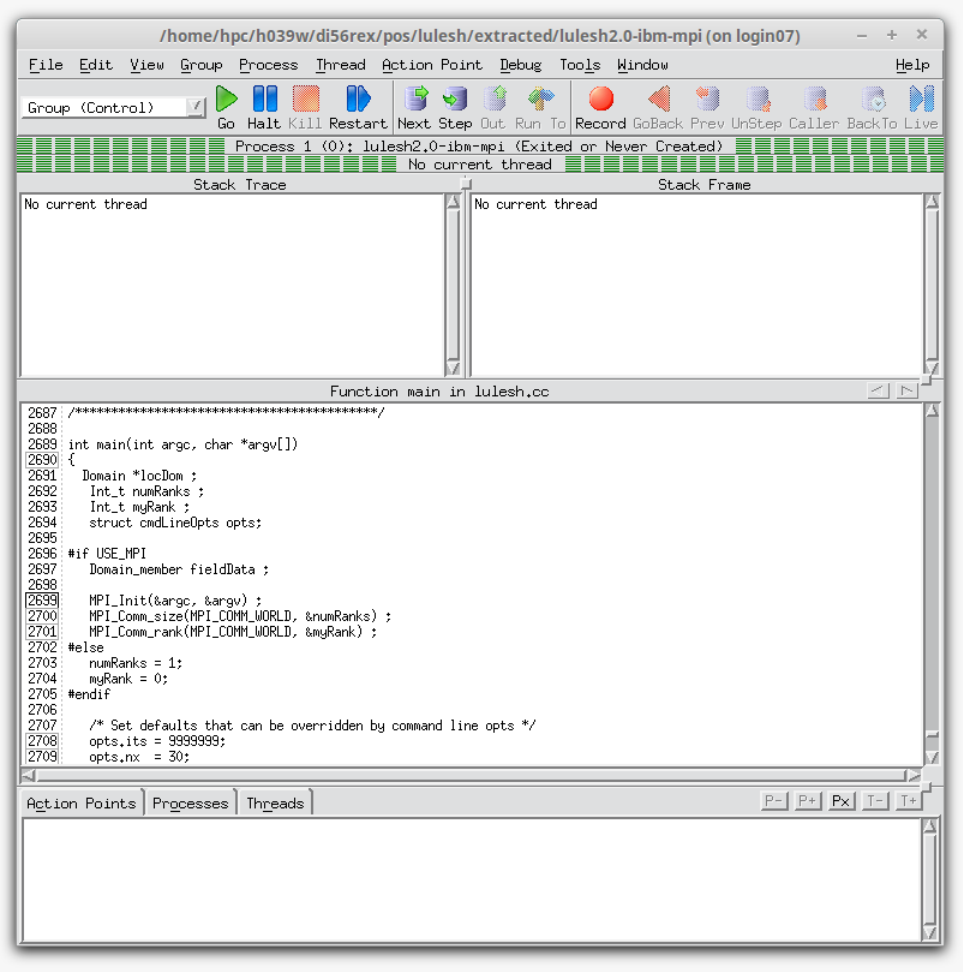


The `ENVIRONMENT` tab allows you to set up any necessary environment variables. Click the `Next` button to continue:



The last tab is just a preview of the launch command. You can now click the `Start Session` button to launch the parallel application.

5 TotalView GUI



In this assignment, we will be working with the GUI version of TotalView.

5.1 Task 2 (20 points total)

Include a brief description of the following aspects of TotalView's GUI in the report:

- Session Manager (3 points)
- Root Window (3 points)
- Process Window (7 points)
  - Stack Trace Pane (1 point)
  - Stack Frame Pane (1 point)
  - Source Pane (1 point)
  - Action Points, Processes, Threads Pane (1 point)
- Variable Window (3 points)

6 TotalView Basic Operations

Make sure that you learn how to do the following tasks from the tutorials and documentation of TotalView:

- Control execution
- Setting breakpoints
- Diving into functions
- View memory (variables and arrays)

6.1 Task 3 (20 points total)

Describe in the report how the above operations are performed in TotalView (each 3 points, 12 total). Give a short explanation on why these operations are important in a debugger (each 2 points, 8 total).

7 MPI with TotalView

MPI applications run with multiple processes and communicate using messages. Each process needs to call `MPI_Init` in order to set up its local data-structures and then be able to send messages and synchronize with other processes.

Launch the MPI version of the benchmark in a TotalView session. Use TotalView's `Source Pane` to set breakpoints at `MPI_Comm_size` and `MPI_Comm_rank`. Use the `Stack Frame Pane` to observe if each process stores the group size and its own rank. Determine when the variables for rank and size change as the MPI application initializes and then sets its control variables. Make sure that each process receives its own unique rank.

Take some time to investigate the dominant routine of the benchmark. Identify one array or vector that is important for the computation. Use TotalView's Visualizer to visualize one or more arrays. Refer to the following page in the tutorial: Visualizing Array Data.

7.1 Task 4 (10 points total)

State the name of the array you decided to visualize and include a snapshot of its visualization in the report (5 points). It is not necessary to understand the actual meaning of the values in the array.

7.1.1 Questions

1. What is the name of the MPI rank variable in the benchmark? (1 point)
2. What is the name of the MPI size variable in the benchmark? (1 point)
3. Where are these variables first set in the benchmark's source code (file name and line number)? (3 points)

8 MPI+OpenMP with TotalView

OpenMP uses a fork-join model of execution. The definitions of parallel regions, parallel loops, etc., are defined via OpenMP pragmas. Make sure to investigate the location of these pragmas in the benchmark.

Set any necessary breakpoints and then launch the MPI+OpenMP binary in its session. You need to make sure to read the documentation and set the environment correctly for the placement of MPI processes and OpenMP threads. Populate as many cores as possible in the two nodes with the combination of processes

and threads, given the benchmark's constraints. The actual number of threads and processes to use is left to each group to decide.

Use TotalView's GUI to iterate through the program and dive into a routine where a parallel region is entered and the new threads are forked by the OpenMP master thread.

8.1 Task 5 (10 points total)

Justify in the report the core and thread combination used (3 points) and explain how thread and process counts are controlled in TotalView (3 points). Include a screen capture of the before and after effect of the fork-join model (3 points), by navigating to a parallel region and looking at the `Threads Pane` in TotalView. Explain what the fork-join model is (1 point).