Programming of Supercomputers

Assignment 2: Parallel Debugging and Performance Analysis

Madhura Kumaraswamy
kumarasw@in.tum.de

Prof. Michael Gerndt
gerndt@in.tum.de

09-11-2018

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. Although scientific software is often parallelized to produce results in less time, the correctness of the solution may be of equal importance as its timeliness.

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. Parallel debuggers come in handy while solving performance issues. In this 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. Hence, 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 users to inspect the source code as well as the state of the memory directly, and provides control of threads. It supports C, C++ and Fortran applications as well as CUDA for NVIDIA GPU kernels in recent versions.

After debugging the code, optimizing the performance is very important for the development of efficient parallel applications. For this, students will use Vampir for performance analysis using traces in the *.otf2* format obtained from Score-P, which is a scalable instrumentation and measurement framework. Vampir is a well established commercial software, and provides an intuitive framework for analysis, which enables developers to view program behavior at any level of detail.

In this assignment, the students will first understand common issues that arise when developing OpenMP and MPI applications. Then, they will learn how to prepare and build a parallel application for analysis with the TotalView debugger. Finally, they will learn how to instrument the code with Score-P and generate traces for visualization using Vampir.

2 Submission Instructions and General Information

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

1. A 5 to 10 page report with the required answers.
   - Submit the report in PDF format.
   - Plots and figures should be used to enhance any explanations.
   - Provide PRECISE answers for each task in the same order as the questions.
   - The report must contain the contribution of each group member to the assignment (max. 2 sentences/member).

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, most desktop and laptop PC hardware, and even mobile devices take advantage of parallel programming.

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

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>@wm.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 2) tempdir/1.0 3) lrztools/1.0 4) intel/17.0 5) mkl/2017 6) poe/1.4 7) mpi.ibm/1.4
8) lrz/default 9) totalview/8.14
```

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 the MPI and MPI+OpenMP binaries. 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 and linker flags are set to:

```
-g -O2
```

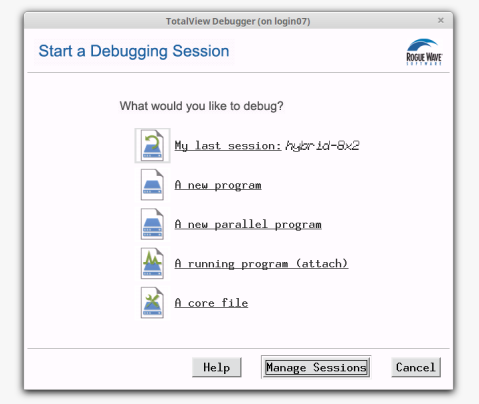
This is specified in the LRZ's TotalView instructions page. Please note that higher levels of optimizations will affect the ability of TotalView to match your source code, so please enable only the flags specified above. Rebuild your benchmark using `make clean && make` so that the new flags are now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

4.1.1 Session Creation

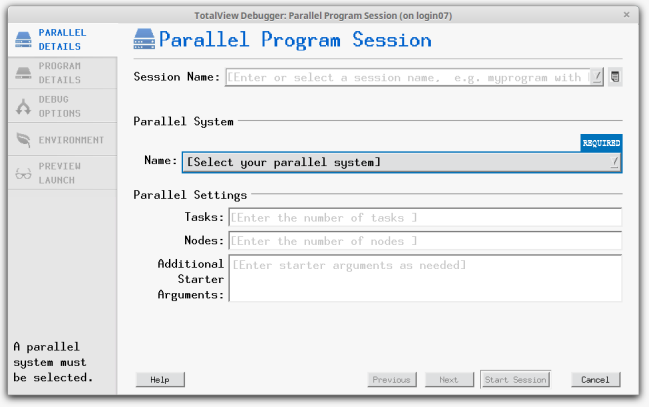
Once you have built the two binaries, make sure that you are in the directory of the benchmark, and launch TotalView:

```
cd <benchmark-directory>
totalview
```

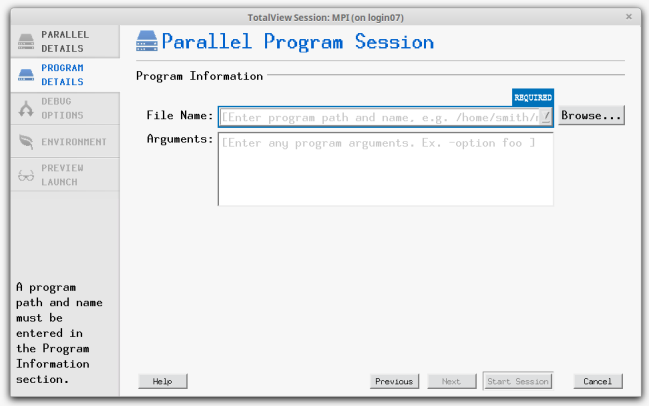
This will start the TotalView GUI.



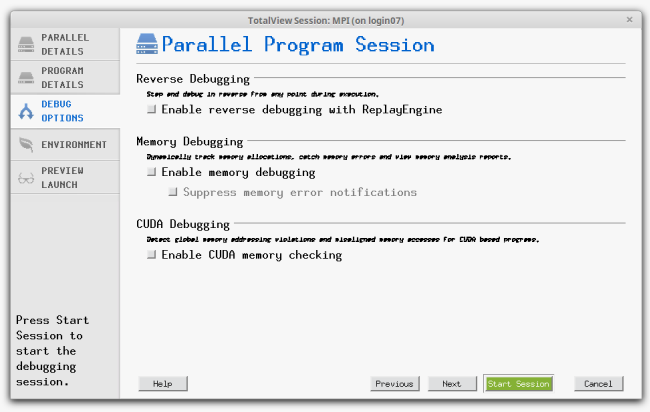
To set up a session, you can go back to the session manager by using the menu `File->Manage Sessions...` from the main TotalView window, or using the welcome screen once you restart TotalView. Select `A new parallel program`. Specify a name for the new session. Select `poe - Linux` as the parallel system. Under `Parallel Settings`, set the `Tasks (-procs)` option to the required number of processes to run the benchmark on 2 nodes. Each of the fat nodes has 40 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.



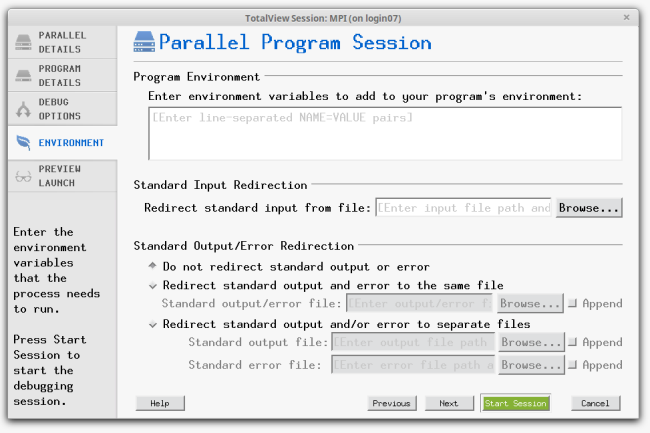
In the `PROGRAM DETAILS` tab, make sure to pick the correct binary for the session by clicking in the `Browse...` button and using the file dialog. For this exercise, the `Arguments` can be left empty.



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.



In the `ENVIRONMENT` tab, you will need to update the environment variables to set the number of threads for OpenMP. Click the `Next` button to continue.



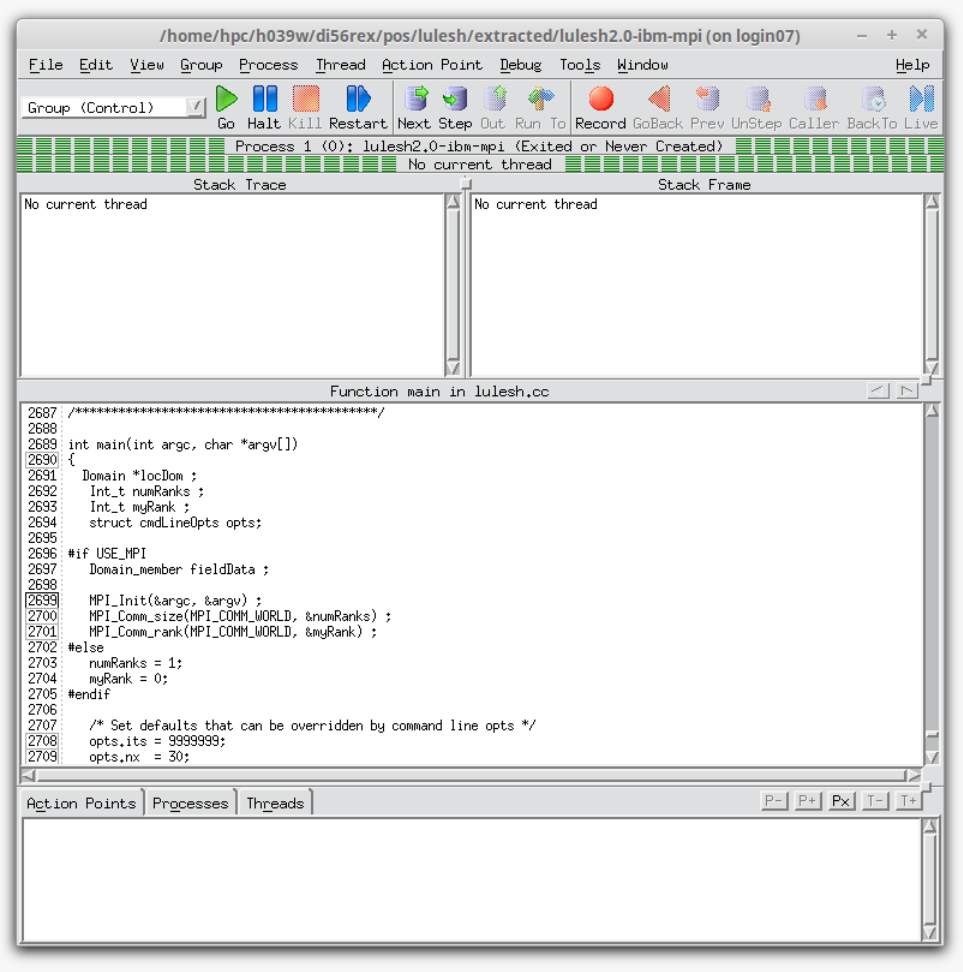
The last tab is just a preview of the launch command. You can now click the `Start Session` button to launch the parallel application.

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

Some of the instructions in the LRZ's website (in the above list) are outdated. The last link contains instructions that are the easiest to follow, and is the most useful for this assignment.

5 TotalView GUI



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

5.1 Task 2 (12 points total)

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

1. Session Manager (2 points)
2. Root Window (2 points)
3. Process Window (2 points)
   - Stack Trace Pane (1 point)
   - Stack Frame Pane (1 point)
   - Source Pane (1 point)
   - Action Points, Processes, Threads Pane (1 point)
4. Variable Window (2 points)

6 Debugging with TotalView

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 (16 points total)

1. Describe in the report how the above operations are performed in TotalView. (2 points each, 8 total)
2. Give a short explanation on why these operations are important in a debugger. (2 points each, 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 (9 points total)

1. State the name of the array you decided to visualize and include a snapshot of its visualization in the report. It is not necessary to understand the actual meaning of the values in the array. (4 points)
2. What is the name of the MPI rank variable in the benchmark? (1 point)
3. What is the name of the MPI size variable in the benchmark? (1 point)
4. 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 (9 points total)

1. Justify in the report the core and thread combination used. (3 points)
2. Explain how thread and process counts are controlled in TotalView. (2 points)
3. Explain what the fork-join model is. (1 point)
4. Include a screen capture of the before and after effect of the fork-join model by navigating to a parallel region and looking at the `Threads Pane` in TotalView. (3 points)

9 Performance Analysis with Vampir

Vampir is a performance analysis and visualization tool that displays dynamic run-time behavior graphically. It provides statistics and performance metrics for different events. It enables users to examine communication patterns, imbalances in the I/O, memory usage or computation, and overheads.

Vampir was developed at the Center for Applied Mathematics of Research Center Jülich and the Center for High Performance Computing of the Technische Universität Dresden. Vampir has been a commercial product since 1996. The Leibniz Compute Centre (LRZ) provides access to Vampir (via a module) to all users of the SuperMUC. Vampir is used to analyze trace log files in the *.otf2* format. Unlike profiling where metrics for events are aggregated over the entire application execution time, tracing records application events as a combination of timestamp, event type, and event specific data. This enables a detailed observation of parallel programs.

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>@wm.supermuc.lrz.de
```

You will need to use the Intel MPI, scorep and vampir/9.0 modules for this assignment. Make sure that you load them in your session:

```
module unload mpi.ibm
module load mpi.intel scorep vampir/9.0
```

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 3) lrztools/1.0 5) mkl/2017 7) lrz/default 9) scorep/3.0
2) tempdir/1.0 4) intel/17.0 6) poe/1.4 8) mpi.intel/2017 10) vampir/9.0
```

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 Score-P and Vampir, you can now proceed to build your benchmark.

9.1 Preparing the Benchmark for Score-P

You will now build the MPI+OpenMP binary for instrumentation with Score-P. Set the following:

```
MPICXX = scorep --mpp=mpi mpiCC -DUSE_MPI=1
CXX = $(MPICXX)
```

Additionally, make sure that the compiler flags are set to:

```
CXXFLAGS = -O3 -qopenmp -I.
```

and the linker flags to:

```
LDFLAGS = -O3 -qopenmp
```

Rebuild your benchmark using `make clean && make` so that the new flags are now in effect.

9.2 Running the benchmark

Download the *scorep.filt* file from moodle and copy it to the benchmark directory on SuperMUC. Alternately, you can create a new *scorep.filt* file on SuperMUC and copy-paste the contents from the original file from moodle.

Create a new LoadLeveler job script as follows:

```
#@ wall_clock_limit = 00:30:00
#@ job_name = pos-lulesh-hybrid
#@ job_type = MPICH
#@ class = fat
#@ output = pos_lulesh_hybrid_$(jobid).out
#@ error = pos_lulesh_hybrid_$(jobid).out
#@ node = 1
#@ total_tasks = 32
#@ node_usage = not_shared
#@ energy_policy_tag = lulesh
#@ minimize_time_to_solution = yes
#@ island_count = 1
#@ queue

module unload mpi.ibm
module load mpi.intel
module load scorep
date
ulimit -c unlimited

export SCOREP_TIMER="clock_gettime"
export SCOREP_ENABLE_PROFILING=false
export SCOREP_ENABLE_TRACING=true
export SCOREP_TOTAL_MEMORY=2GB
export SCOREP_FILTERING_FILE=scorep.filt

module list

export OMP_NUM_THREADS=4

mpiexec -n 8 ./lulesh2.0 -i 20 -s 20 -p

date
exit
```

Submit the job. After the job has completed, a folder with the name `scorep-<timestamp_identifier>`, containing the *traces.otf2* file will be generated. Open the trace file using Vampir, as shown below:

```
vampir scorep-<timestamp_identifier>/traces.otf2
```

You will get a view similar to the image below:

Have a look at the Vampir user manual to understand the information that is displayed in each chart in Vampir. The guide also provides use-cases and hints on how to look at performance bottlenecks.

10 MPI+OpenMP with Vampir

10.1 Task 6 (26 points total)

1. What are the main differences in the way events are recorded by gprof and Score-P? (3 points)
2. Does Vampir support post-mortem or online analysis? Explain. (2 points)
3. Explain the communication bottlenecks that occur in point-to-point (4 points) and collective MPI communication (4 points). For example, late-sender, ...
4. Take a screen capture of one of the MPI communication bottlenecks from the above question, and describe the situation. (3 points)
5. How does the *Performance Radar* help in analysis? (3 points)
6. Describe the MPI message pattern in the *Communication Matrix* tab. (2 points)
7. Take a screen capture of a fork-join, and describe the situation. (2 points)
8. Do you observe any performance bottlenecks from Section 3.1 (Task 1) in the code? (3 points)