Prof. Dr. M. Bader
S. Rettenberger M.Sc.
C. Ferreira M.Sc.

TUM, SCCS (I5)
Scientific Computing:
High Performance Computing

November 23, 2015

This assignment covers profilers and performance analysis tools.

Literature

- *GNU gprof*:
  `https://sourceware.org/binutils/docs/gprof/`
- *LRZ: Intel VTune*:
  `https://www.lrz.de/services/software/programmierung/intel_amplifier/`
- *Scalasca User Manual*:
  `http://apps.fz-juelich.de/scalasca/releases/scalasca/2.2/docs/UserGuide.pdf`
- *Score-P User Manual*:
  `https://silc.zih.tu-dresden.de/scorep-current/pdf/scorep.pdf`
- *Perf Wiki*:
  `https://perf.wiki.kernel.org/index.php/Main_Page`

1 Gauss elimination – GNU gprof

GNU gprof can be used to visualize profile data. It works very closely together with the compiler flag "`-pg`" from GNU and Intel compiler. If the option `-pg` is added to the compiler AND linker flags, the execution of the program will generate a profile data file `gmon.out`. This file can be analyzed using the gprof tool.

In this exercise, we will revisit the Gauss elimination from the first worksheet. First, you will need to increase the number of systems of equations to get reasonable profile data. Make sure that the execution is at least several seconds. Recompile the program with `-pg` and analyze the profile data with gprof. Compare the results from the GNU and Intel compiler.

1. Why do the interesting functions `init` and `gauss_elimination` do not show up in the call graph when the Intel compiler is used? Try to solve this by adding additional compiler options or minimal code modifications.
2. Which function should be optimized first (according to the profile data)?
3. Compare the runtime of the application with and without profiling. Modify the functions and/or introduce new functions such that the code is better suitable for profiling with GNU gprof. The profile data should still show the difference in the runtime of initialization and the actual Gauss elimination.

2 Quicksort – Intel VTune Amplifier XE

Intel VTune Amplifier XE is a profiler from Intel. It is available on the Linux Cluster and can be loaded with `module load amplifier_xe`. VTune comes with a graphical user interface (`amplxe-gui`) and a command line tool (`amplxe-cl`).

In this exercise we will optimize the parallel Quicksort from the second worksheet using VTune. The goal is, to find an optimal `final`-clause for the OpenMP tasks.

Setup a project for Quicksort with the VTune GUI and configure a “Basic Hotspots” analysis. However, do not start the analysis from the GUI! If you do this, the application will run on the login node. As a workaround, the GUI can export the command to run the analysis with `amplxe-cl`. Put the command in a SLRUM script and submit it. Once the application is finished, you can visualize the results of the analysis with the GUI.

1. Find the optimal `final`-clause for your OpenMP parallelization of Quicksort using Intel VTune Amplifier XE.
2. Look at the other analysis types besides “Basic Hotspots”. Are they useful for the Quicksort application? What is the difference between them?
3. Does the optimal final clause depend on the number of threads or the array length?
4. **Hint:** Use X forwarding in SSH to start the VTune GUI directly on the cluster.

3 CG – Scalasca

Scalasca is another tool that allows to gather profile data. It works very similar to GNU gprof but comes with advanced features and more flexibility. On the Linux Cluster Scalasca is available via `module load scalasca/2.2.2`.

In this exercise we will look at the MPI parallelization of the conjugate gradient method from the third worksheet. We will use Scalasca to find load imbalances in the parallelization. To instrument the code, you have to place the prefix `scalasca -instrument [options]` before the compiler and linker, e.g. `scalasca -instrument mpicxx poisson.cpp`.

With the `options` you can control which parts of the code you want to instrument. Start with `--mpp=mpi` to instrument normal functions (default) and MPI functions. The instrumented application will generate profile data in a folder called `scorep-*`. The data can be visualized with `cube`. Besides the graphical interface there is also a set of command line tools available to convert or dump the profile data which might be useful for automatic processing.

A nice feature of Scalasca is, that it does not only support automatic but also manual code instrumentation. The manual instrumentation can be enabled with the option `--user`. To define a “region” in your code, you can use the macros `SCOREP_USER_REGION_DEFINE( handle )`, `SCOREP_USER_REGION_BEGIN( handle, "foo", SCOREP_USER_REGION_TYPE_COMMON )` and `SCOREP_USER_REGION_END( handle )` defined in `scorep/SCOREP_User.h`. You will see your defined regions similar to normal functions in `cube`.

1. Profile your parallel CG implementation. Do you detect load imbalances for certain numbers of tasks? Explain the box plot in `cube`.

2. The automatic instrumentation usually generates too much profile data. Disable it with `--nocompiler` and enable the manual instrumentation. Define suitable regions in the code to get detailed information without to much overhead.
   Hint: You can use the header from the SeisSol project to easily compile with and w/o manual instrumentation. The header defines dummy macros if manual instrumentation is not enabled:
   `https://github.com/SeisSol/SeisSol/blob/master/src/Monitoring/instrumentation.fpp`

3. Parallelize the important loops in the application with OpenMP. Does the hybrid MPI/OpenMP parallelization give you a speedup? Besides manual, compiler and MPI instrumentation, Scalasca can also instrument OpenMP pragmas with `--thread=omp`. Does the OpenMP instrumentation help you optimizing your program?

4 DGEMM – Perf

Perf is a Linux tool to gather hardware performance counters for an application. In contrast to many profilers, it does not instrument the code but relies on performance counters from the CPU and the kernel. Thus, it will not detect how long a code spends in a specific function but hardware and software events such as page faults, cache misses or branch miss predictions. We will use Perf to analyze the DGEMM code from the first worksheet.

1. Use `perf list` to get a list of all available events. Which events might be the most useful for the DGEMM optimization?

2. Run `perf stat -e <event> <command>` to measure the cache misses for your DGEMM code. Measure the cache misses from the reference code and your optimized version. Measure different cache levels!

3. Compare the results from Perf with Valgrind. Why are they different?

4. Analyze your OpenMP parallel DGEMM from the second worksheet with Perf. Find suitable performance counters for the parallel DGEMM and describe there meaning.

5. **Hint:** Perf is currently not available on the Linux Cluster. Use the MAC-Cluster for this exercise: `http://www.mac.tum.de/wiki/index.php/MAC_Cluster`

Deliverables

The following deliverables have to be handed in no later than 08:00 AM, Monday, 7th December, 2015. Small files (<1 MB in total) can be send as an attachment directly to *rettenbs AT in.tum.de*, larger files have to be uploaded at a place of your choice, e.g.

`https://github.com/`, `http://home.in.tum.de/`, `https://www.dropbox.com`. In either case inform us about the final state of your solution via e-mail.

- All of your code.
- Slides for the presentation during the next meeting; remember to address all questions in the tasks
- Output of all runs. Figures (e.g. scaling graphs) if applicable.
- Documentation how to build and use your code.
- Output/Screenshots from the profilers.