UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

INF-3201 – MPI performance analysis

Edvard Pedersen
Edvard.Pedersen@uit.no

Performance analysis approaches

- Instrumenting code
  - Add code at different parts to measure how long the code takes to execute
- Profiling
  - Sample what code is running at intervals
- Tracing
  - Get a timeline view of code executing

Instrumenting code

- Implement timing code to find out how long different stages of the application takes to run
- Already implemented in the code you have been given
- However, to get detailed data, a lot of instrumentation calls have to be inserted

Profiling

- Inspect the execution e.g. 100 times/second, see which code is being executed at that time
- Gives a view of where time is spent
- Can (possibly) give an overview of execution, usually with some caveats

Tracing

- Record entry and exit times for function calls (usually automatically)
- Gives an extremely detailed timeline view of the program execution
- Potentially huge performance hit

What is the difficult part?

- None of these things look the same across all threads/nodes
  - What is true on compute-3-2 is not always true on compute-5-1
- What happens in one thread has an effect on what happens in the others
  - Speedup of 1000x in one thread may only lead to a 1.05x speedup of the program
- Getting an overview of the execution across many threads and nodes can be difficult

What can we do?

- What the run.sh script does:
  - Profiling with gprof/gperf done per-node
  - Collect tracing data with OpenMPI's tracing support
- Gprof output is by default aggregated, and doesn't include MPI calls (no symbol information in the OpenMPI installation)
  - Less useful for profiling the MPI part of the program
  - Gperf includes the MPI calls, if you want to examine it all in detail
- The run.sh script creates a summary of the tracing data, but it should be visualized in a real trace visualizer
  - Ravel is installed on the cluster
  - Can also use Vampir (vampir.eu) on your own machine

How to interpret the data

- Gprof
- Gperf
- Otfprofile
- Ravel

How to use ravel

- Add «source /share/apps/etc/uvrocks-env» to ~/.bashrc on uvrocks
- SSH tunnel from localhost:something to uvrocks.cs.uit.no:(5900 + display number)
- Start VNC server on uvrocks on the chosen display
- Connect to localhost:something with a VNC viewer locally
- Start Ravel
- Import your .otf file

What to put in the report

- «As we can see from the tracing results, the worload is balanced and 10% of the execution time is spent communicating»
- «I chose to paralellize function X, since profiling shows that 95% of the runtime is spent in this function»
- «Profiling shows that optimization Y increases computation time by Z, while reducing communcation time by A, leading to a speedup of B»