Debuggers and Performance Tools

June 2016 | Markus Geimer
Outline

Make it work, make it right, make it fast.

*Kent Beck*

Port to Jureca

**Debuggers:**
- TotalView
- DDT
- MUST
- ...

**Performance Tools:**
- Performance Reports
- Score-P
- Scalasca
- Vampir
- TAU
- ...

---

M. Geimer

JURECA Porting & Tuning Workshop, June 2016
Debugging Tools
Parallel Debugger

- UNIX Symbolic Debugger for C, C++, F77, F90, PGI HPF, assembler programs
- "Standard" debugger
- Special, non-traditional features
  - Multi-process and multi-threaded
  - C++ support (templates, inheritance, inline functions)
  - F90 support (user types, pointers, modules)
  - 1D + 2D Array Data visualization
  - Support for parallel debugging (MPI: automatic attach, message queues, OpenMP, pthreads)
  - Scripting and batch debugging
  - Memory Debugging
  - CUDA and OpenACC support
- http://www.roguewave.com
- **NOTE:** License limited to 2048 processes (shared between all users)
TotalView: Main Window

Stack trace

Toolbar for common options

Local variables for selected stack frame

Source code window

Break points

Stack Trace

Function main in hello-mpi.c

```c
#include <stdio.h>
#include <mpi.h>
int main(int argc, char *argv[]) {
    int ierr, myrank, numprocs;
    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    printf("hello from %d of %d\n", myrank, numprocs);
    return 0;
}
```
DDT Parallel Debugger

- UNIX Graphical Debugger for C, C++, F77, F90 programs
- Modern, easy-to-use debugger
- Special, non-traditional features
  - Multi-process and multi-threaded
  - 1D + 2D array data visualization
  - Support for MPI parallel debugging (automatic attach, message queues)
  - Support for OpenMP (Version 2.x and later)
  - Support for CUDA and OpenACC
  - Job submission from within debugger

- http://www.allinea.com
- NOTE: License limited to 64 processes (shared between all users)
DDT: Main Window

- **Process controls**
- **Process groups**
- **Source code**
- **Variables**
- **Expression evaluator**
- **Stack trace**
MUST

- Next generation MPI correctness and portability checker
- http://doc.itc.rwth-aachen.de/display/CCP/Project+MUST

MUST reports
- Errors: violations of the MPI-standard
- Warnings: unusual behavior or possible problems
- Notes: harmless but remarkable behavior
- Further: potential deadlock detection
Memory and Threading Error Checkers

- Can detect errors such as
  - Memory leaks
  - Memory corruption
  - Allocation/deallocation API mismatches
  - Illegal memory accesses
  - Data races
  - Deadlocks

- Available on Jureca:
  - Valgrind
  - Intel Inspector
Performance Analysis Tools
Allinea Performance Reports

- **Single page** report provides quick overview of performance issues
- Works on unmodified, optimized executables
- Shows CPU, memory, network, and I/O utilization

- Supports MPI, multi-threading, and accelerators
- Saves data in HTML, CVS, or text form

- **Note:** License limited to 512 processes (with unlimited number of threads)
Example Performance Reports

Summary: cp2k.popt is CPU-bound in this configuration

The total wallclock time was spent as follows:

- **CPU** 56.5%  
  - Time spent running application code. High values are usually good.  
  - This is **average**, check the CPU performance section for optimization advice.

- **MPI** 43.5%  
  - Time spent in MPI calls. High values are usually bad.  
  - This is **average**; check the MPI breakdown for advice on reducing it.

- **I/O** 0.0%  
  - Time spent in filesystem I/O. High values are usually bad.  
  - This is **negligible**; there's no need to investigate I/O performance.

This application run was **CPU-bound**. A breakdown of this time and advice for investigating further is in the CPU section below.

**CPU**

A breakdown of how the 56.5% total CPU time was spent:

- Scalar numeric ops 27.7%  
- Vector numeric ops 11.3%  
- Memory accesses 60.9%  
- Other 0.0%

The per-core performance is **memory-bound**. Use a profiler to identify time-consuming loops and check their cache performance.

Little time is spent in **vectorized instructions**. Check the compiler's vectorization advice to see why key loops could not be vectorized.

**MPI**

Of the 43.5% total time spent in MPI calls:

- Time in collective calls 8.2%  
- Time in point-to-point calls 91.8%  
- Estimated collective rate 169 Mb/s  
- Estimated point-to-point rate 50.6 Mb/s

The point-to-point transfer rate is low. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait. Use an MPI profiler to identify the problematic calls and ranks.

**I/O**

A breakdown of how the 0.0% total I/O time was spent:

- Time in reads 0.0%  
- Time in writes 0.0%  
- Estimated read rate 0 bytes/s  
- Estimated write rate 0 bytes/s

No time is spent in I/O operations. There's nothing to optimize here!

**Memory**

Per-process memory usage may also affect scaling:

- Mean process memory usage 82.5 Mb  
- Peak process memory usage 89.3 Mb  
- Peak node memory usage 7.4%

The peak node memory usage is low. You may be able to reduce the total number of CPU hours used by running with fewer MPI processes and more data on each process.
Score-P

- Community instrumentation and measurement infrastructure
  - Developed by a consortium of performance tool groups
  
  - Common data formats improve tool interoperability
  
  - Supports C, C++, and Fortran using MPI, SHMEM, OpenMP, POSIX threads, CUDA, OpenCL and combinations

- Highly configurable:
  - Basic and advanced profile generation
  - Event trace recording
  - Using instrumentation and sampling (experimental)

http://www.score-p.org
Call-path Profile: Example
Collection of trace-based performance analysis tools

- Specifically designed for large-scale systems
- Unique features:
  - Scalable, automated search for event patterns representing inefficient behavior
  - Scalable identification of the critical execution path
  - Delay / root-cause analysis
- Based on Score-P for instrumentation and measurement
  - Includes convenience / post-processing commands providing added value
- http://www.scalasca.org
Example: Wait-state, Critical Path & Delay Analysis

- Searches for *wait states* in communication & synchronization
- Determines a profile of the application’s *critical path*
- Identifies *delays* as the root causes of wait states
Scalasca Trace Analysis Example

- Additional wait-state metrics from the trace analysis
- Delay / root-cause metrics
- Critical-path profile
Vampir Event Trace Visualizer

- **Offline** trace visualization for Score-P’s OTF2 trace files
- **Visualization of MPI, OpenMP** and application events:
  - **All** diagrams highly customizable (through context menus)
  - Large variety of displays for **ANY** part of the trace
- [http://www.vampir.eu](http://www.vampir.eu)

- **Advantage:**
  - Detailed view of dynamic application behavior
- **Disadvantage:**
  - Requires event traces (huge amount of data)
  - Completely manual analysis
Vampir Displays

Number of Messages

Communication Matrix View

Process 0. Values of Counter "MEM_APP_ALLOC" over Time

50 M

0 M

Function Summary

Process Summary

Context View

Property | Value
--- | ---
Display | Function Summary
Function Group | MPI (6)
Accumulated Exclusive Time | 748.945947 s (29.193292%)
TAU

- Very portable tool set for instrumentation, measurement and analysis of parallel multi-threaded applications
- http://tau.uoregon.edu/

- Supports
  - Various profiling modes and tracing
  - Various forms of code instrumentation
  - C, C++, Fortran, Java, Python
  - MPI, multi-threading (OpenMP, Pthreads, …)
TAU: Basic Profile View
Multi-platform sampling-based call-path profiler
Works on unmodified, optimized executables
http://hpctoolkit.org

Advantages:
- Overhead can be easily controlled via sampling interval
- Advantageous for complex C++ codes with many small functions
- Loop-level analysis (sometimes even individual source lines)
- Supports POSIX threads

Disadvantages:
- Statistical approach that might miss details
- MPI/OpenMP time displayed as low-level system calls
Example: hpcviewer

```
Example: hpcviewer

Callpath to hotspot
```

```
associated source code
```

```
Example: hpcviewer
```
Further tools

- **Darshan**
  - I/O characterization tool logging parallel application file access
  - Shows counts of file access operations, times for key operations, histograms of accesses, etc.

- **Intel Advisor**
  - Vectorization optimization

- **Intel Trace Analyzer and Collector**
  - Graphical tool for understanding MPI application behavior

- **NVIDIA Visual Profiler**
  - GPU performance analysis
  - Supports CUDA and OpenACC
A combination of different methods, tools and techniques is typically needed!

- Analysis
  - Statistics, visualization, automatic analysis, data mining, ...
- Measurement
  - Sampling / instrumentation, profiling / tracing, ...
- Instrumentation
  - Source code / binary, manual / automatic, ...
Pointers

- **VI-HPS** (Virtual Institute - High-Productivity Supercomputing)
  - International collaboration between tool development groups
  - Organizes many tool trainings
    - Full/half-day tutorials at conferences
    - Multi-day “bring-your-own-code” tuning workshops
  - [http://www.vi-hps.org](http://www.vi-hps.org)

- **POP** (Performance Optimization and Productivity)
  - EU Centre of Excellence providing performance optimization services
  - [http://pop-coe.eu](http://pop-coe.eu)