Linaro Forge

Debugging and Optimization Tools for HPC
HPC Development Solutions from Linaro

Best in class commercially supported tools for HPC

Linaro Forge

Performance Engineering for any architecture, at any scale
Supported Platforms

- Intel Compiler
- ROCm
- CCE
- ACfL
- GCC
- NVHPC
- IBM XL
- Compiler
- Python

- Intel MPI
- HPE MPI
- MPICH
- Open MPI
- IBM Spectrum MPI
- Slurm
- PALS
- CPU Architecture

- RHEL 7+
- SLES 15
- Ubuntu 20.04+
- macOS
- Windows

- AMD ROCm
- NVIDIA CUDA
- GP-GPU Accelerator

- AMD/Intel (x86-64)
- Arm (AArch64)
- Power8 (ppc64le)
- CPU Architecture

Linaro Forge
Debugging with DDT

An interoperable toolkit for debugging

The de-facto standard for HPC development
- Most widely-used debugger in HPC
- Fully supported by Linaro on Intel, AMD, Arm, Nvidia, AMD GPUs, etc.

State-of-the art debugging capabilities
- Powerful and in-depth error detection mechanisms (including memory debugging)
- Available at any scale (from serial to exascale applications)

Easy to use by everyone
- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users
Linaro DDT Debugger Highlights

- The scalable print alternative
- Stop on variable change
- Static analysis warnings on code errors
- Detect read/write beyond array bounds
- Detect stale memory allocations
Multi-dimensional Array Viewer

What does your data look like at runtime?

View arrays
- On a single process
- Or distributed on many ranks

Use metavariables to browse the array
- Example: $i$ and $j$
- Metavariables are unrelated to the variables in your program
- The bounds to view can be specified
- Visualise draws a 3D representation of the array

Data can also be filtered
- “Only show if”: $\text{value}>0$ for example $\text{value}$ being a specific element of the array
Debug Example - Fortran code
Linaro Performance tools

Characterize and understand the performance of HPC application runs

Gather a rich set of data
- Analyses metric around CPU, memory, IO, hardware counters, etc.
- Possibility for users to add their own metrics

Build a culture of application performance & efficiency awareness
- Analyses data and reports the information that matters to users
- Provides simple guidance to help improve workloads’ efficiency

Adds value to typical users’ workflows
- Define application behaviour and performance expectations
- Integrate outputs to various systems for validation (eg. continuous integration)
- Can be automated completely (no user intervention)
MAP Capabilities

MAP is a sampling based scalable profiler
- Built on same framework as DDT
- Parallel support for MPI, OpenMP, CUDA
- Designed for C/C++/Fortran

Designed for ‘hot-spot’ analysis
- Stack traces
- Augmented with performance metrics

Adaptive sampling rate
- Throws data away - 1,000 samples per process
- Low overhead, scalable and small file size
Linaro MAP Source Code Profiler Highlights

- Find the peak memory use
- Fix an MPI imbalance
- Remove I/O bottleneck
- Make sure OpenMP regions make sense
- Improve memory access
- Custom Metrics
MAP Example - ROCm AMD GPU

Profile
- Ran for 6s, taking 300 samples per process
- Able to bring up metadata of the profile
- Mixed CPU [green] / GPU [purple] application
- CPU time waiting for GPU Kernels [purple]
- GPU Kernels graph indicating Kernel activity

GUI information
- GUI is consistent across platforms
- Zoom into main thread activity
- Ranked by highest contributors to app time
Performance Reports Example - Python

Report information
- Summary of application performance
- Actionable insight
- Could integrate into CI/CD

Generating a report
- Able to generate from a MAP file
- Generated through the command-line
- Text output or HTML report

Summary: python-profiling.py is Python Interpreter-bound in this configuration

<table>
<thead>
<tr>
<th>Component</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute</td>
<td>46.7%</td>
</tr>
<tr>
<td>MPI</td>
<td>0.0%</td>
</tr>
<tr>
<td>I/O</td>
<td>0.0%</td>
</tr>
<tr>
<td>Python Interpreter</td>
<td>53.3%</td>
</tr>
</tbody>
</table>

This application run was Python Interpreter-bound.
As very little time is spent in MPI calls, this code may also benefit from running at larger scales.
Running on Frontier

Forge is already on Frontier
module load forge

DDT offline debugging
ddt --offline srun -A <project_id> -t 00:05:00 -p <partition> -N 2 -n 4 --ntasks-per-node=2 ./app

MAP offline profile
map --profile srun -A <project_id> -t 00:05:00 -p <partition> -N 2 -n 4 --ntasks-per-node=2 ./app

Performance report
perf-report srun -A <project_id> -t 00:05:00 -p <partition> -N 2 -n 4 --ntasks-per-node=2 ./app

For remote connection
https://docs.linaroforge.com/latest/html/forge/forge/connecting_to_a_remote_system/connecting_remotely.html
Thank you

Go to www.linaroforge.com
rudy.shand@linaro.org