Arm Forge
Debugging and Optimization Tools for HPC

Beau Paisley<Beau.Paisley@arm.com>
Arm Forge
An interoperable toolkit for debugging and profiling

The de-facto standard for HPC development
- Most widely-used debugging and profiling suite in HPC
- Fully supported by Arm on Intel, AMD, Arm, IBM Power, Nvidia GPUs, etc.

State-of-the art debugging and profiling capabilities
- Powerful and in-depth error detection mechanisms (including memory debugging)
- Sampling-based profiler to identify and understand bottlenecks
- Available at any scale (from serial to petaflopic applications)

Easy to use by everyone
- Unique capabilities to simplify remote interactive sessions
- Innovative approach to present quintessential information to users
HPC Development Solutions from Arm

Best in class commercially supported tools for Linux and high-performance computing

Performance Engineering

for any architecture, at any scale

arm FORGE

Debugger
Profiler
Reporting
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
9 Step guide: optimizing high performance applications

Improving the efficiency of your parallel software holds the key to solving more complex research problems faster. This pragmatic, 9 Step best practice guide will help you identify and focus on application readiness, bottlenecks and optimizations one step at a time.

1. Bugs
- Correct application.

2. Analyze before you optimize
- Measure all performance aspects.
- You can't fix what you can't see.
- Prefer real workloads over artificial tests.

3. Communication
- Track communication performance.
- Discover which communication calls are slow and why.

4. Workload
- Detect issues with balance.
- Slow communication calls and processes.
- Dive into partitioning code.

5. I/O
- Discover lines of code spending a long time in I/O.
- Trace and debug slow access patterns.

6. Memory
- Reveal lines of code bottlenecked by memory access times.
- Trace allocation and use of hot data structures.

7. Cores
- Discover synchronization overhead and core utilization.
- Synchronization-heavy code and implicit barriers are revealed.

8. Vectorization
- Understand numerical intensity and vectorization level.
- Hot loops, unvectorized code and GPU performance revealed.

9. Verification
- Validate corrections and optimal Performance.

Key:
- arm PERFORMANCE REPORTS
- arm FORGE
Arm Performance Reports

No source code needed

Less than 5% runtime overhead

Fully scalable

Run regularly – or in regression tests

Explicit and usable output
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
- Restructure for vectorization
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
Python Profiling

- 19.0 adds support for Python
  - Call stacks
  - Time in interpreter

- Works with MPI4PY
  - Usual MAP metrics

- Source code view
  - Mixed language support

Note: Green as operation is on numpy array, so backed by C routine, not Python (which would be pink)

map --profile jsrun -n 2 python3 ./diffusion-fv-2d.py
WFH Technology, ... Remote Connect

https://developer.arm.com/docs/101136/latest/arm-forge/connecting-to-a-remote-system
‘WFH Technology’, … Offline Debugging

https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/debugging-while-you-sleep
https://community.arm.com/developer/tools-software/hpc/b/hpc-blog/posts/more-debugging-while-you-sleep-with-ddt