Debugging and Profiling with Arm Tools

ryan.hulguin@arm.com
Agenda

- Introduction to Arm Tools
- Remote Client Setup
- Debugging with Arm DDT
- Other Debugging Tools
- Break
- Examples with DDT
- Lunch
- Profiling with Arm MAP
- Examples with MAP
- Obtaining Support
Introduction to Arm HPC Tools
Arm Forge
An interoperable toolkit for debugging and profiling

- The de-facto standard for HPC development
  - Available on the vast majority of the Top500 machines in the world
  - Fully supported by Arm on x86, IBM Power, Nvidia GPUs and Arm v8-A.

- 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
Arm Performance Reports

Characterize and understand the performance of HPC application runs

Gathers a rich set of data
- Analyses metrics 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 (e.g. continuous integration)
  - Can be automated completely (no user intervention)
Software tools-centric view

- Analyze (Arm Performance Reports)
- Debugging (Arm DDT)
- Perf Optimization (Arm MAP)

Demand for software efficiency

Demand for developer efficiency

Open interfaces (e.g. JSON APIs)

Continuous integration (e.g. Jenkins, etc.)

Version Control (e.g. Git, etc.)

DB

NEW VERSION

Debug/optimize, edit, commit, build, repeat
Using Forge and the remote client
Different ways to run Arm Forge...

Here
(remote launch + reverse connect)

There
(interactive mode + reverse connect)

There
(offline OR interactive mode)

Ultimately, that’s where the tools will run. But what about the GUI?
Forge Remote Client

• The latest version of Forge can be downloaded from https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge

• It is important to have the remote client version match what is installed on the system
Forge Remote Client

Debugging with Arm DDT
Print statement debugging

• The first debugger: print statements
  • Each process prints a message or value at defined locations
  • Diagnose the problem from evidence and intuition

• A long slow process
  • Analogous to bisection root finding

• Broken at modest scale
  • Too much output – too many log files
Typical types of bugs

- **BOHR BUG**
  • Steady and dependable, I’ll be there for you.

- **HEISEN BUG**
  • Oh, you are debugging? Let me hide for a sec!

- **MANDEL BUG**
  • Chaos is my name and you shall fear me.

- **SCHRODIN BUG**
  • I am buggy **AND** not buggy. How about that?
Debugging by discipline

Debugging a problem is much easier when you can:

- **Make and undo changes fearlessly**
  - Use a **source control** (CVS, ...)

- **Track what you’ve tried so far**
  - Write **logbooks**

- **Reproduce bugs with a single command**
  - Create and use **test script**

$ mkdir logs
$ vi logs/segfault-at-4096-procs

When running `lu.E.4096` with the `trace-4410.dat` set, the job exited with: `'An error occurred in MPI_Send [11346-209:25319] on communicator MPI_COMM_WORLD MPI_ERR_RANK: invalid rank'`.

To reproduce: `apiexec -n 4096 lu.E.4096 trace-4410.dat` on `supernuc`. Seems to happen every time.

- Tried reading core file with `gdb` "File truncated"
- Set `ulimit -c unlimited` and ran again: ...

$ logs/segfault-at-4096-procs.sh
Sep 27 15:29: Queued as job.43214
Sep 27 18:01: Running...
Sep 27 19:29: FAIL
Arm DDT – the debugger

Who had a rogue behaviour?
- Merges stacks from processes and threads

Where did it happen?
- Leaps to source

How did it happen?
- Diagnostic messages
- Some faults evident instantly from source

Why did it happen?
- Unique “Smart Highlighting”
- Sparklines comparing data across processes
Arm DDT cheat sheet

Load the environment module (on Cori/Edison)
• $ module load allinea-forg

Prepare the code
• $ cc -O0 -g myapp.c -o myapp.exe

Start Allinea DDT in interactive mode
• $ ddt srun -n 8 ./myapp.exe arg1 arg2

Or use the reverse connect mechanism
• On the login node:
  • $ ddt &
  • (or use the remote client)
  • Then, edit the job script to run the following command and submit:
    • ddt --connect srun -n 8 ./myapp.exe arg1 arg2
Examples
Example Files

• Once connected to cori, download the examples to your home directory

• `cp /project/projectdirs/training/DebugProfile_201804/NERSC_Training.tar.gz ~/`
DDT Demonstration
Exercise:
Fixing a simple crash
Algorithm: $C = A \times B + C$

1- Master initialises matrices $A$, $B$ & $C$
2- Master slices the matrices $A$ & $C$, sends them to slaves
3- Master and Slaves perform the multiplication
4- Slaves send their results back to Master
5- Master writes the result Matrix $C$ in an output file
Fix a simple crash in a MPI code

Objectives:
- Discover Arm DDT's interface
- Debug a simple crash in a MPI application interactively
- Use the tool in a cluster environment

Key commands:
- Compile the application: $ make
- Clean and recompile for debugging: $ make clean && make DEBUG=1
- Use the debugger with reverse connect
- Accept the incoming connection!
- Can you find out and fix the bug?
Exercise: Identifying Out-of-Memory Accesses
Critical memory crash

Objectives:
- Use the memory debugging feature
- Diagnose and fix a memory problem

Key commands:
- Compile the application with debugging flags: `$ make`
- Recompile using the memory debugging library (statically link through Makefile LFLAGS)
- Enable memory debugging in the “Run window”
- Change the amount of checks, enable guard pages
- Can you see the memory issue can you fix it?
Exercise:
Understanding hangs
Deadlock

Objectives:
- Witness a deadlock and attach to the running processes
- Use Arm DDT Stack feature
- Use Arm DDT evaluation window

Key commands:
- Compile with: $ make
- Submit the job to run the application with 10 processes: it works.
- Run it again with 8 processes: it hangs!
- Leave the application run in the queue and attach to it with the debugger
- OR (if attaching is not supported) Submit the job again with the debugger
- Observe where it hangs. Can you fix the problem?
Exercise:
Detecting memory leaks
Memory leaks

Objectives:

- Use Arm DDT’s offline mode
- Use the memory debugging feature
- Diagnose and fix a memory leak problem

Key commands:

- Compile the application for debugging
  
  `$ make`
- Edit a job script to use the debugger in offline mode with memory debugging on and submit the job
- Open the resulting *.html file
- Can you see the memory leak?
- Restart the debugger in interactive mode. Can you see any hint from the debugger?
Profiling with Arm MAP
The complete HPC developer workflow

- System access made simple
  - Work remotely or locally
  - Same full capabilities

- Be confident changes work
  - Re-use Scheduler reservation ...
  - ... Edit
  - ... Build
  - ... Test
  - Commit
Why profiling?

How to improve the performance of an application?

Profiling: a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.  

(Wikipedia)

How?

– Select representative test case(s)
– Profile
– Analyse and find bottlenecks
– Optimise
– Profile again to check performance results and iterate
How to profile?

Different methods

• Tracing
  – Records and timestamps all operations
  – Intrusive

• Instrumenting
  – Add instructions in the source code to collect data
  – Intrusive

• Sampling
  – Automatically collect data
  – Not intrusive
Some types of profiles

**Hotspot**
- One function corresponds to more 80% of the runtime
- Large speed-up potential
- Best optimisation scenario

**Spike**
- The application spends most of the time in a few functions
- Speed-up potential depends on the aggregated time
- Variable optimisation time

**Flat**
- Runtime split evenly between numerous functions, each one with a very small runtime
- Little speed-up potential without algorithmic changes
- Worst optimisation scenario
Arm MAP: Performance made easy

Low overhead measurement
- Accurate, non-intrusive application performance profiling
- Seamless – no recompilation or relinking required

Easy to use
- Source code viewer pinpoints bottleneck locations
- Zoom in to explore iterations, functions and loops

Deep
- Measures CPU, communication, I/O and memory to identify problem causes
- Identifies vectorization and cache performance
Arm MAP cheat sheet

Load the environment module
- $ module load allinea-forg

Prepare the code
- $ cc -O3 -g myapp.c -o myapp.exe

Edit the job script to run Arm MAP in “profile” mode
- $ map --profile srun ./myapp.exe arg1 arg2

Open the results
- On the login node:
  - $ map myapp_Xp_Yn_YYYY-MM-DD_HH-MM.map
  - (or load the corresponding file using the remote client connected to the remote system or locally)
Typical memory hierarchy

- **Registers**
  - Size (bytes): 192
  - Latency from next level (cycles): 4

- **L1 Cache**
  - Size (bytes): 32k
  - Latency from next level (cycles): 12

- **L2 Cache**
  - Size (bytes): 256k
  - Latency from next level (cycles): 26

- **L3 Cache**
  - Size (bytes): 2M
  - Latency from next level (cycles): 230-360

- **Main memory**
  - Size (bytes): 2G
  - Latency from next level (cycles): ?

Example of Intel Sandy Bridge
Speeding up memory accesses

High performance is possible when:

• There is an opportunity for cache re-use
• Data is local to the core for quick usage
• CPU gets data from memory to cache before it is actually needed
Memory access patterns

Data locality

- Temporal locality: use of data within a short time of its last use
- Spatial locality: use memory references close to memory already referenced

**Temporal locality example**
```
for (i=0 ; i < N; i++) {
    for (loop=0; loop < 10; loop++) {
        ... = ... x[i] ...
    }
}
```

**Spatial locality example**
```
for (i=0 ; i < N*s; i+=s) {
    ... = ... x[i] ...
}
```
Memory Accesses and Cache Misses

```c
for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        A[i*n+j]=...
    }
}
```

i=0, n=4

HIT

```c
for(i=0; i<n; i++) {
    for(j=0; j<n; j++) {
        A[j*n+i]=...
    }
}
```

i=0, n=4

MISS
Exercise:
Optimizing memory accesses
Resolve high memory accesses issues

Objectives:
- Discover Arm MAP’s interface
- Profile the MPI matrix multiplication example and find out the performance issue
- Use the tool in a cluster environment

Key commands:
- Compile the application
  $ make
- $ map --profile srun myApp.exe
- Open the result in the GUI on the login node once the job has completed
  $ map *.map
- What is the bottleneck of the application? Can you identify performance problems?
Resolving workload imbalances
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. Memory
   - Reveal lines of code bottlenecked by memory access times.
   - Trace allocation and use of hot data structures.

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

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

8. Verification
   - Validate corrections and optimal Performance.

Key:
- ✔ arm PERFORMANCE REPORTS
- ✔ arm FORGE
Load balancing in theory

Balancing the workload is critical because:
  – Processors may be idle for an extended period of time
  – They could have been doing some work instead of burning energy

Examples of load balancing
  • *Owner computes*
    Balance done through data distribution
  • *Independent tasks*
    Balance done through prediction/statistics
  • *A mix of various components*
    Balance between scalar workload and communications (for instance)
Redistributing the workload

Several techniques exist to balance the workload

• “Simple” redistribution of data
• Dynamic balancing using space filling curves

Example

Step 1: Adaptive Refinement of a domain in subsequent levels
Redistributing the workload

Example

Step 2: Load distribution of an adaptively refined domain

Step 3: Space Filling Curve
Load balancing can be counter intuitive

There is an asymmetry between processors having too much work and having not enough work. It is better to have one processor that finishes a task early than having one that is overloaded so that all others wait for it.

Corollary:

When it comes to load balancing, the “costliest” function shown by the profiler is not the bottleneck. The bottleneck is the “cheapest” one.

Workload imbalance webinar video
https://youtu.be/MScwYTNGOp0
Exercise:
Improving IOs
Detect workload imbalance and optimise IO

Objectives:
- Exhibit the workload imbalance in the code (on 1 or 2 nodes)
- Make suggestions to fix the problem

Key commands:
- Compile the application
  $ make
- $ map --profile srun -n 8 ./myApp.exe
- Open the profiling results in the GUI on the login node once the job has completed
  $ map *.map
- How can you fix the imbalance problem?
Maximizing application efficiency with Performance Reports
Arm Performance Reports benefits

- Reduced run time
- Higher throughput

Constraints:
- Energy
- Processor
- Storage
- Networks

Analytics

Arm Performance Reports

Benefits
“Learn” with Arm Performance Reports

Very simple start-up
No source code needed
Fully scalable, very low overhead
Rich set of metrics
Powerful data analysis
Metrics overview

Multi-threaded parallelism

SIMD parallelism

Load imbalance

OMP efficiency

System usage
Arm Performance Reports cheat sheet

Load the environment module

- $ module load allinea-reports

Edit the job script to prefix the mpirun command

- perf-report srun -n 8 ./myapp.exe

Analyse the results

- $ cat myapp_8p_1n_YYYY-MM-DD_HH:MM.txt
- $ firefox myapp_8p_1n_YYYY-MM-DD_HH:MM.html
Exercise:
Maximizing scientific output
Maximise efficiency

Objectives:

• Generate a performance report of a simple code
• Find the best parameters to maximize the application efficiency
  – Compilation flags
  – Number of processes
  – Number of nodes

Key commands:

• Compile:
  $ make
• $ perf-report srun -n 8 ./myapp.exe
Forge User Guide

- Online documentation is always available at https://developer.arm.com/products/software-development-tools/hpc/documentation
- Direct link to DDT User Guide https://developer.arm.com/docs/101136/latest/ddt
- Local user guide is available in your Forge installation /path/to/arm/forge/doc/userguide-forge.pdf
Obtaining Support
Obtaining Support

• For simple queries, use the web form at https://www.arm.com/products/development-tools/hpc-tools/contact-support

• For more advanced issues, email support-hpc-sw@arm.com
  This allows you attach screenshots, source code, and debug log files
Debug Log Files
Debug Log Files

• In the event that DDT crashes or does not work like expected, a debug log file will be helpful to the arm support team
• Debug log files can be generated by passing arguments to DDT
• For Example:

  ddt --debug --log=crash.log aprun -n 16 ./myProgram.exe
Q&A
Thank You
Danke
Merci
谢谢
ありがとう
Gracias
Kiitos
감사합니다
धन्यवाद
תודה