Arm Tools Workshop

Nick Forrington <nick.forrington@arm.com> 14th September 2018

Confidential © 2018 Arm Limited

Orm

Agenda

- 9:00 Introduction
- 9:30 Remote Client Setup
- 9:45 DDT Getting Started
- 10:30 15-minute break
- 10:45 Offline Debugging
- 11:15 Memory Debugging Leaks and Errors
- 12:00 Lunch
- 13:00 Performance Reports and MAP
- 14:30 15-minute break
- 14:45 GPU Debugging and Profiling
- 16:00 Discussion / Finish

Performance Engineering Methodology and Tools

arm

Confidential © 2018 Arm Limited

Welcome to the age of machine-scale computing

It's dangerous to go alone! Take this.

30 years ago: human-scale computing



Cray 2:

- 4 vector processors
- 1.9 gigaflops (9.5 mflops/Watt)

Today: machine-scale computing



Summit:

- 2,282,544 cores
- 2,000,000 gigaflops (154 mflops/Watt)

Your brain is no longer enough

No way around it, you need tools to achieve maximum performance.

- Supercomputers are now incomprehensibly complex.
- Naïve optimization may harm performance.
- **Performance engineering tools are essential** for realizing performance at scale.





Your brain is no longer enough

No way around it, you need tools to achieve maximum performance.

- Supercomputers are now incomprehensibly complex.
- Naïve optimization may harm performance.
- **Performance engineering tools are essential** for realizing performance at scale.



Arm's solution for any architecture, at any scale

Commercial tools for aarch64, x86_64, ppc64le and accelerators



Arm's solution for any architecture, at any scale

Commercial tools for aarch64, x86_64, ppc64 and accelerators



arm

Arm Forge = DDT + MAP

An interoperable toolkit for debugging and profiling



Commercially supported by Arm





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, 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

DDT: Production-scale debugging

Isolate and investigate faults at scale

- Which MPI rank misbehaved?
 - Merge stacks from processes and threads
 - Sparklines comparing data across processes
- What source locations are related to the problem?
 - Integrated source code editor
 - Dynamic data structure visualization
- How did it happen?
 - Parse diagnostic messages
 - Trace variables through execution
- Why did it happen?
 - Unique "Smart Highlighting"
 - Experiment with variable values

Stacks (AII)	
Processes	Function
150120	⊨_start
150120	Ėlibc_start_main
150120	🖻 main
150120	Ėpop (POP.f90:81)
150120	📥 initialize_pop (initial.f90:119)
150120	📄 📩 init_communicate (communicate.f90:87)
150119	create_ocn_communicator (communicate.f90:300)
	create_ocn_communicator (communicate.f90:303)

Locals	Current Line(s)	Cur	rent Stack	
Current Line	e(s)			8 ×
Variable N	ame		Value	
jcol			36	
mype			"l n 1 2724	
	Current Line Variable N	Current Line(s) Variable Name <i>jcol</i>	Current Line(s) Variable Name <i>jcol</i>	Current Line(s) Variable Name Value <i>jcol</i> 36

DDT: Feature Highlights



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": \$value > 0 for example \$value being a specific element of the array

Multi-Dimensional Array Viewer		×
ray Expression: [tables[\$i][\$j]	*	Evaluate
stributed Array Dimensions: None 🚔 How do I view distributed arrays?		Cancel
Staggered Array What does this do?		✓ Align Stack Frames
Range of \$j		Auto-update
Fr <u>o</u> m: 0 + Fr <u>o</u> m: 0 +		
Display: Rows 💠 Display: Columns 💠		
Only show if		
Only show if: See Examples		
ata Ta <u>b</u> le <u>S</u> tatistics		
🔶 Goto 🔌 Visualize 💾 Export 🖸 Full Window		
j		
0 1 2 3 4 5 6 7 8 9 10 11 i 0 1 2 3 4 5 6 7 8 9 10 11	2	
	24	
2 3 6 9 12 15 18 3 4 8 12 16 20 24		
4 6 12 16 20 24 4 5 10 15 20 25 30		1000
5 6 12 18 24 30 36		200
6 7 14 21 28 35 42 7 8 16 24 32 40 48		
8 9 18 27 36 45 54	S	
9 10 20 30 40 50 60	S	III
10 11 22 33 44 55 66 rows		1111 Million
Help		
		45 column

MAP: Production-scale application profiling

Identify bottlenecks and rewrite code for better performance

- Run with the representative workload you started with
- Measure all performance aspects with Arm Forge Professional

Examples:

\$> map -profile aprun -n 8 ./example
\$> map -profile jsrun -n 6 ./example

Profiled: clover_leaf on 32 processes, 4 nodes, 32 cores (1 per process) Sampled from: Wed Nov 9 2016 15:28:37 (UTC) for 309.1s Hide Metrics					
	an and an all the annual second and the formation from the second second second second second second	e na pharpacilite a , a corpet d'an ait Athronisticean a praisi			
Iterations / s 8.0 328.% 0 Grind time 1.76.%		and the second			
Step time 0.00 s		Zoom #J 📰 💿			
	a 0.2 %, OpenMP 80.0 %, MPI 19.7 %, OpenMP overhead 0.1 %, Sleeping 0.1 %				
		Time spent on line 75 @ 8			
51.2%	ALL flux_calc() ALL advection() ALL reset_field()	Breakdown of the 51.2% time spent on this line: Executing instructions 0.0% Calling other functions 100.0%			
Input/Output Project Files OpenMP Stacks	OpenMP Regions Functions				
OpenMP Stacks		0 8			
	Function(s) on line	1			
51.2% 8.4% <0.1% 22.9% 0.2% -0.1%	e / clover jeat ehydro *advection_module_advection i timasten_module_timasten				

Profiled: My_code.exe on 64 proc	esses Started: Fri Sep 20 14:59:09	9 2013 Runtime: 35s Time in MPI: 45 %	Hide Metrics
Memory usage (M) 9.4 - 777.9 (454.6 av)		
MPI call duration (ms) 0 - 5,575.1 (341.0 av)		
CPU floating-point (%) 0 - 90 (8.2 avg)		: Mean MPI call duration 341.0 ms; Mean CPU floating-point 8.2 %;	Metrics, Reset
14:53:03-14:53:44 (range 54.)	ros): Mean Memory usage 404.0 M	; Mean MPI Call duration 341.0 ms; Mean CPU floating-point 6.2 %;	metrics, meser
T My_code.f90 🔀			
		tationn	
	100	MODULE EXCITATION	ha Ma Ang
	102 ! 103 ₪ module deriva	tivee	ng mig mig mig mig mig mig mig mig mig mi
	140 !		
	141 !	MAIN CODE	
	143 🖂 program Vel_V	ort_3D_FP	
	144 use data_mc	***:	
	145 use wall_exci 146 implicit none		
	147 include 'mpif		
	148 double precis	<pre>ion :: max_omx_dt,max_omy_dt,max_omz_dt,t,time_cal</pre>	
		<pre>tion,i,j,k,nn,fwcnt,count_max,counter,ios,next_file_at,W_cnt(1:4)</pre>	
	150 character*30 151	<pre>:: str,file_type,str_t,num_2_str</pre>	
0.1%	152 call MPI INIT	(ierr)	
		SIZE(MPI_COMM_WORLD, npro, ierr)	
nput/Output Project Files	arallel Stack View		
allel Stack View	1		
tal Time	MPI Function(s) on line	Source	Position
	vel vort 3d fp . <un< td=""><td>program Vel Vort 3D_FP</td><td>My code.f90:143</td></un<>	program Vel Vort 3D_FP	My code.f90:143
63.0%	31.4% Etime_integration	call time_integration	My_code.f90:330
16.9% mml	5.3% mod_rank_read_file 6.3% velocity solver	<pre>call mod_rank_read_file_all_its_own(str,nn,ios) ! Restart from last check call velocity solver</pre>	point My_code.f90:297 My_code.f90:337
1.8%	€ <unknown></unknown>	<pre><uklobicity_solver <="" <uklobicity_solver="" uklobicity_solver="" ul=""></uklobicity_solver></pre>	My_code.190:337
1.5%	1.4% 🖲 vel_vort_3d_fp_	call cell_identifier	My_code.f90:190
4.1%	1.3% 🖲 91 others		

How MAP is different

MAP's flagship feature is lightweight, highly scalable performance profiling



Arm Performance Reports

Characterize and understand the performance of HPC application runs



Commercially supported by Arm



Accurate and astute insight



Relevant advice to avoid pitfalls

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)

Arm Performance Reports

A high-level view of application performance with "plain English" insights

arm Performance	Command:	mpiexec.hydra -host node-1,node-2 -map-by socket -n 16 -ppn 8 ./Bin/low_freq///Src//hydro -i ./Bin/low_freq////Input/input_250x125_corner.nml	I/O A breakdown of the 16.2% I/O time:			
REPORTS	Resources: Memory:	2 nodes (8 physical, 8 logical cores per node) 15 GiB per node	Time in reads	0.0%		
	Tasks:	16 processes, OMP_NUM_THREADS was 1	Time in writes	100.0%		
	Machine: Start time:	node–1 Thu Jul 9 2015 10:32:13	Effective process read rate	0.00 bytes/s		
	Total time: Full path:	165 seconds (about 3 minutes) Bin//Src	Effective process write rate	1.38 MB/s		
Summarv	: hvdro i	is MPI-bound in this configuration	effective transfer rate. This	write operations with a very low may be caused by contention for the ess patterns. Use an I/O profiler to s are affected.		



Time spent running application code. High values are usually good. This is **very low**; focus on improving MPI or I/O performance first Time spent in MPI calls. High values are usually bad. This is **high**; check the MPI breakdown for advice on reducing it

Time spent in filesystem I/O. High values are usually bad. This is **average**; check the I/O breakdown section for optimization advice

Arm Performance Reports Metrics

Lowers expertise requirements by explaining everything in detail right in the report.



Forge and Performance Reports at ORNL

- Machines
 - Titan
 - Summit
 - Wombat
 - Your laptop
 - ...
- User Guides
 - <u>https://www.olcf.ornl.gov/software_package/forge/</u>
 - <u>https://www.olcf.ornl.gov/software_package/arm-performance-reports/</u>





arm

Arm Forge Quick Start

Tool cheat sheets

Confidential © 2018 Arm Limited



Arm DDT cheat sheet

Start DDT interactively, remotely, or from a batch script.

- Load the environment module:
 - \$ module load forge
- Prepare the code:
 - \$ cc -OO -g myapp.c -o myapp.exe
 - \$ ftn -00 -g myapp.f -o myapp.exe
- Start DDT in interactive mode:
 - \$ ddt aprun -n 8 ./myapp.exe arg1 arg2 ...
- Or use reverse connect:
 - Connect the remote client (or launch "ddt" on the login node)
 - Run the follow command, or edit a job script and submit:
 - \$ ddt --connect aprun -n 8 ./myapp.exe arg1 arg2 ...
- Offline mode
 - **\$ ddt --offline** aprum -n 8 ./myapp.exe arg1 arg2 ... (see ddt --help for more options)

Arm MAP cheat sheet

Generate profiles and view offline

- Load the environment module
 - \$ module load forge
- Prepare the code
 - \$ cc -O3 ... -g myapp.c -o myapp.exe
 - \$ ftn -O3 ... -g myapp.f -o myapp.exe
- Interactive (Collect and View)
 - \$ map aprun -n8 ./myapp.exe arg1 arg2
- Offline: edit the job script to run Arm MAP in "profile" mode
 - \$ map --profile aprun -n8 ./myapp.exe arg1 arg2
- View profile in MAP:
 - On the login node:
 - \$ map myapp_Xp_Yn_YYY-MM-DD_HH-MM.map
 - (or load the corresponding file using the remote client connected to the remote system or locally)

Arm Performance Reports cheat sheet

Generate text and HTML reports from application runs or MAP files

- Load the environment module:
 - \$ module load perf-reports
- No need to prepare application
- Run the application:
 - perf-report aprun -n 8 ./myapp.exe
- ... or, if you already have a MAP file:
 - **perf-report** myapp_8p_1n_YYYY-MM-DD_HH:MM.txt
- Analyze the results
 - \$ cat myapp_8p_1n_YYYY-MM-DD_HH:MM.txt
 - \$ firefox myapp_8p_1n_YYYY-MM-DD_HH:MM.html

Forge Remote Client

Confidential © 2018 Arm Limited



The Forge GUI and where to run it

DDT and MAP provide powerful GUIs that can be run in a variety of configurations.



After connecting the client

Three options to proceed

Click run and launch via the the GUI

- Works well simple jobs
- DDT can launch a batch job for you
- Can be tricky to replicate complicated launch environments or flags

Edit a batch script to use ddt --connect

- Best option for complex batch scripts
- Also for long running non-interactive jobs
- . \$MODULESHOME/init/bash module load forge ddt --connect aprun ...

Use ddt --connect from an interactive session

 Useful if you want to try many runs within different launch options/environments

Launching the Forge Remote Client

The remote client is a stand-alone application that runs on your local system

Install the Arm Remote Client (Linux, macOS, Windows)

- https://developer.arm.com/products/software-development-tools/hpc/downloads/download-arm-forge
 - Searching for "Arm Forge Download" will typically take you here
- https://www.olcf.ornl.gov/tutorials/forge-remote-client-setup-and-usage/

Connect to the cluster with the remote client

- Open Forge Remote Client
- Create a new connection: Remote Launch → Configure → Add
 - Hostname: <username>@titan.ccs.ornl.gov
 - Remote installation directory: /sw/xk6/forge/18.2.2/sles11_binary
 - You can also get the above path by: module load forge/18.2.2; echo \$DDT_HOME
- Connect!
- Training material: ~nforr/training/arm-tools-workshop.tar.gz

Working with the queue

- Connect the remote client
- In a terminal, SSH to Titan and launch and interactive session
 - •qsub -I -A <account> -q debug -l
 nodes=1,walltime=01:00:00
- module load forge/18.2.2
- Launch aprun command prefixed with ddt --connect

DDT Getting Started

Crash and hang

Confidential © 2018 Arm Limited

arm

$\mathbf{C} = \mathbf{A} \times \mathbf{B} + \mathbf{C}$

Simply multiply and add two matrices

Algorithm

- 1. Rank 0 (R0) initialises matrices A, B & C
- 2. R0 slices the matrices A & C and sends them to Rank 1...N (R1+)
- 3. R0 and R1+ perform the multiplication
- 4. R1+ send their results back to R0
- 5. R0 writes the result matrix C to file



Fix a simple crash in a MPI code

Simple matrix multiply and add? No problem! Except that it crashes...

Exercise Outline

- Objectives
 - Discover Arm DDT's interface
 - Interactively debug a crash in a MPI application

Commands

- \$ make
- \$ aprun -n 4 ./mmult1_c.exe
- # Observe crash
- \$ ddt --connect ./mmult1_c.exe
- # Observe cause of crash

Initial Result: Crash!

🔵 💿 johlin02@johlin02-VM: ~/MUG18/01_walkthrough/1_crash johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash\$ make mpicc -q -ffast-math -OO -DDEBUG -std=c99 mmult1.c -o mmult1 c.exe -lm mpif90 -g -ffast-math -O0 -DDEBUG -cpp mmult1.f90 -o mmult1_f90.exe -lm johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash\$ mpirun -np 4 ./mmult1 c.exe 0: Size of the matrices: 64x64 0: Initializing matrices... 0: Sending matrices... 0: Processina... [johlin02-VM:mpi rank 0][error sighandler] Caught error: Segmentation fault (signal 11) 3: Receiving matrices... 2: Receiving matrices... 1: Receiving matrices... 2: Processing... [johlin02-VM:mpi rank 2][error sighandler] Caught error: Segmentation fault (signal 11) 1: Processing... [johlin02-VM:mpi rank 1][error sighandler] Caught error: Segmentation fault (signal 11) BAD TERMINATION OF ONE OF YOUR APPLICATION PROCESSES PID 9160 RUNNING AT johlin02-VM EXIT CODE: 139 CLEANING UP REMAINING PROCESSES YOU CAN IGNORE THE BELOW CLEANUP MESSAGES

YOUR APPLICATION TERMINATED WITH THE EXIT STRING: Segmentation fault (signal 11) This typically refers to a problem with your application. Please see the FAQ page for debugging suggestions johlin02@johlin02-VM:~/MUG18/01_walkthrough/1_crash\$ \$

Answer: Fix incorrect limits on k-loop

Incorrect limits lead to invalid memory access

Before

164	<pre>do i=0,size/nslices-1</pre>
165	do j=0,size-1
166	res=0.0
167	<pre>do k=size,size*size</pre>
168	res=A(i*size+k)*B(k*size+j)+res
169	end do
170	C(i*size+j)=res+C(i*size+j)
171	end do
172	end do

After

164	<pre>do i=0,size/nslices-1</pre>
165	do j=0,size-1
166	res=0.0
167	do k=0,size-1
168	res=A(i*size+k)*B(k*size+j)+res
169	end do
170	C(i*size+j)=res+C(i*size+j)
171	end do
172	end do



Problem #2

Fixing the crash reveals another issue

- Run the program again, and found out why the program now hangs
- Either launch again with DDT
- Or launch without, and attach
 - Ensure your nodes file is set to \$DDT_HOME/titan.nodes in the options dialog
 - Click attach, from the welcome page. This will may result in SSH prompts as DDT scans the other Titan login/batch nodes, before detecting your job
 - Alternatively, launch: ddt --connect --attachmpi=<aprun-pid>

Program now hangs

_				
	0:	Size of the matrices:	64 x	64
	0:	Initializing matrices		
	1 :	Receiving matrices		
	2:	Receiving matrices		
	3:	Receiving matrices		
	0:	Sending matrices		
	1 :	Processing		
	0:	Processing		
	2:	Processing		
	1 :	Sending result matrix		
	2:	Sending result matrix		
	0:	Receiving result matrix		

Answer: Fix incorrect limits on i-loop

Incorrect limits on i-loop lead to unmatched MPI_Send

Before

73 do i=1,nproc-2

- 75 call MPI_Send(mat_b, size*size, & MPI_DOUBLE, i, 200+i, & MPI_COMM_WORLD, ierr)
- 76 call MPI_Send(mat_c(slice*i), slice, & MPI_DOUBLE, i, 300+i, & MPI_COMM_WORLD, ierr)

77 end do

After

73	do i=1,nproc-1
74	<pre>call MPI_Send(mat_a(slice*i), slice, & MPI_DOUBLE, i, 100+i, & MPI_COMM_WORLD, ierr)</pre>
75	call MPI_Send(mat_b, size*size, & MPI_DOUBLE, i, <mark>200+</mark> i, & MPI_COMM_WORLD, ierr)
76	<pre>call MPI_Send(mat_c(slice*i), slice, & MPI_DOUBLE, i, 300+i, & MPI_COMM_WORLD, ierr)</pre>

77 end do

Offline Debugging

arm

Confidential © 2018 Arm Limited

Run DDT in offline mode

Run the application under DDT and halt or report when a failure occurs.

- You can run the debugger in non-interactive mode
 - For long-running jobs
 - For automated testing, continuous integration...
- To do so, use the following arguments:
 - \$ ddt --offline --output=report.html aprun ./myapp.exe
 - --offline enable non-interactive debugging
 - -output specifies the name and output of the non-interactive debugging session
 - Html
 - Txt
 - Add --mem-debug to enable memory debugging and memory leak detection
 - Add --break-at=<location> to report stacks and variables at certain locations
 - Add --trace-at=<location>,variable1,variable2 to evaluate variables/expressions at certain locations
 - See --help for more information

Offline Log

Snippet from an earlier crash

Process stopped in mmult (mmult1.f90:168) with signal SIGSEGV (Segmentation fault). Reason/Origin: address not mapped to object (attempt to access invalid address)

Additional Information

▼ Stacks

Processes	Function		Source			Variables
	mmult2 (mmult1.f90:92)	call	<pre>mmult(size, nproc, mat_a, mat_b, mat_c)</pre>	🕨 Ran	k 0,	thread 1
	mmult (mmult1.f90:168)	▼ res=A(i*size+k)*B(k*size+j)+res		🔻 Ran	thread 1	
				Nam	e	Value
		165.	do j=0,size-1 res=0.0	a	<	<aggregate value=""></aggregate>
				b	<	<pre><aggregate value=""></aggregate></pre>
		167.	do k=size,size*size	С	<	<pre><aggregate value=""></aggregate></pre>
		168.	res=A(i*size+k)*B(k*size+j)+res	i		<u> </u>
		169.	end do	1		
		170.	C(i*size+j)=res+C(i*size+j)]]		0
		171.	end do	k	-	_ 260 (from 260 to 262)
				nslic	ces -	<u> </u>
				res	-	5380641 (from 4189752 to 13189176)
				size	-	<u> </u>
When to use offline debugging

- If you're not available
 - e.g. when you have a long wait in the queue
- Scriptable
 - Debug many jobs
 - Nightly builds / Continuous integration

Memory Debugging

Allocation tracking and guard pages

arm

Confidential © 2018 Arm Limited

DDT's heap memory debugging framework

On Titan, we need to link DDT's memory debugging libraries

- Caveat: Does not work with PGI and Fortran
- Handled by helper module loaded after the forge module
- module load forge/18.2.2; module load ddt-memdebug

Other systems (including Summit)

- No linking required for dynamically linked binaries (handled by LD_PRELOAD)
- For static binaries, check the Forge user guide

Run	
Run: mpirun -n 8 ./mmult2_c.exe	Details
Command: mpirun -n 8 ./mmult2_c.exe	
OpenMP	Details
CUDA: Track allocations: enabled, Detect invalid accesses: disabled	Details
♂ Track GPU allocations (also enables CPU memory debugging)	
Detect invalid accesses (memcheck)	
Memory Debugging: Fast, 1 guard page after, Backtraces, Preload	Details
Plugins: none	Details
Help Options	<u>R</u> un Quit

When manual linking is used, untick "Preload" box

<u> P</u> reload the men ■	nory debugging library <mark>L</mark> ang	guage: C++, thread	s 🗧
	only works for programs link atically linked, you must rel		
Fast	Balanced	Thorough	Custom
<u>Enabled</u> Checks:	basic	More Ir	nformation
Heap O <u>v</u> erflow/Ur	nderflow Detection		
⊠ <u>A</u> dd guard page	es to detect out of bounds he	eap access	
Guard pages: 1	Add guard pages: Aft	er 🗧	
Ad <u>v</u> anced			
□ <u>C</u> heck heap cor	nsistency every 100 🛔 he	ap operations	
✓ Store stack <u>b</u> ac	ktraces for memory allocati	ons	
□ <u>O</u> nly enable for	these processes:		
0	100% Se	lect All x2 x0.	5 1%
Help		ОК	Cancel

Three levels of heap debugging overhead



basic

• Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)

check-fence

•Check the end of an allocation has not been overwritten when it is freed.

free-protect

• Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.

Added goodness

•Memory usage, statistics, etc.

Balanced

•Overwrite the bytes of freed memory with a known value.

alloc-blank

free-blank

•Initialise the bytes of new allocations with a known value.

check-heap

•Check for heap corruption (e.g. due to writes to invalid memory addresses).

realloc-copy

 Always copy data to a new pointer when reallocating a memory allocation (e.g. due to realloc)

Thorough

•Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.

check-blank

check-funcs

•Check the arguments of addition functions (mostly string operations) for invalid pointers.

See user-guide: Chapter 12.3.2

arm

Tri-diagonal solve: segmentation fault

Crashing with invalid memory reference. Sounds like a job for a memory debugger!

Exercise Outline

- Objectives
 - Use DDT's memory debugging features
 - Use guard pages to find out-of-bounds access
- First lets run without DDT
 - \$ module swap PrgEnv-pgi PrgEnv-gnu
 - \$ make
 - \$ aprun -n 4 ./trisol.exe
- Now let's see where it crashes in DDT (without memory debugging)
 - \$ ddt --connect aprun -n 4
 ./trisol.exe

Invalid memory access

	2. ssh
lustre/atlas/scratch/nforr/stf010 forr@titan-login2:/lustre/atlas/scratch/nforr/stf010\$ aprun	-n4 /ccs/home/nforr/training/arm-tools-workshop/memory-error-trisolve/trisol.exe
<pre>** glibc detected *** *** glibc detected *** /ccs/home/nfor ve/trisol.exe: : double free or corruption (lprev)double fr ***</pre>	r/training/am=tools-workshop/memory-error-trisolve/trisol.exe/cs/home/nfarr/training/anm=tools-workshop/memory-error-t ee or corruption (lprev): 0x: 0x0000000000000000000000000000000
** glibc detected *** *** glibc detected *** /ccs/home/nfor ve/trisol.exe: : double free or corruption (!prev)double fr	r/training/arm-tools-workshop/memory-error-trisolve/trisol.exe/ccs/home/nforr/training/arm-tools-workshop/memory-error-t ee or corruption (lprev): 0x: 0x0000000000005c520000000000000000000000
*** Backtrace:	
0x614aa4]	
0x619917]	
0x4026dd Backtrace:	
0. F0. C111100.110	
0x[0x614aa4400d46 Backtrace:	
0x[0x614aa44010e0]	
0x[0x[0x6199176199175ee561 Backtrace:	
0x[0x614aa44026dd]	
0x[0x]	
19917[@x]	
88446[0x]	
0x[0x4026dd4010e0]	
826dd488755[0x]	
0x400d465ee561]	
8x Memory map:	
810e0[0x]	
0x[0x400d464007555ee561]	
8x Memory map:	
0x4010e0400755]	
0400000-00739000 r-xp 00000000 00:0e 654725	/var/opt/cray/alps/spool/18792538/trisol.exe
0939000-0093d000 rwop 00339000 00:0e 654725 093d000-0099c000 rwop 00000000 00:00 0	/var/apt/cray/alps/spool/18792538/trisol.exe [heap]
099c000-009fb000 mkp 00000000 00:00 0	(heap)
aaaaaab0000-2aaaaaaaaac000 r-xp 00000000 00:00 0	[vdso]
0] 1:bash 2:bash 3:[tmux]* 4:bash-	"titan-ext1" 22:10 13-Se

Let's try memory debugging

Relink

- module load ddt-memdebug
- make clean; make
- ddt --connect aprun -n 4
 ./trisol.exe
- Launch without guard pages enabled and "Fast" heap debugging.
- The program seems to run fine now why?

And launch in DDT

Application: /home/johlin02/MUG18/03	mem_debugging/trisol.exe
rguments:	▼
stdin file:	
Morking Directory:	😝 🗊 Memory Debugging Options
MPI: 4 processes, MVAPICH 2	✓ Preload the memory debugging library Language: C++, threads
umber of Processes: 4	Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library
mplementation: MVAPICH 2 Change	manually.
npirun arguments	Heap Debugging
OpenMP	Fast Balanced Thorough Custom
CUDA	Enabled Checks: basic More Information
Memory Debugging: Fast, No guard	pages, Heap Overflow/Underflow Detection
Submit to Queue	Add guard pages to detect out of bounds heap access
Environment Variables: none	Guard pages: 1 Add guard pages: After
Plugins: none	
	Advanced
	□ <u>Check heap consistency every</u> 100 <u>+</u> heap ope <u>r</u> ations
	✓ Store stack <u>b</u> acktraces for memory allocations
	Only enable for these processes:
Help Options	0:3 100% Select All x2 x0.5 1%



Guard pages (aka "electric fences")



- A powerful feature...:
 - Forbids read/write on guard pages throughout the whole execution

(because it overrides C Standard Memory Management library)

- ... to be used carefully:
 - Kernel limitation: up to 32k guard pages max ("mprotect fails" error)
 - Beware the additional memory usage cost

OK, this time enable guard pages

The code appears to run fine when launched from the debugger! Why?

Add one guard page after every allocation

😣 🗈 Memory De	ebugging Options			
✓ Preload the mem	ory debugging library <u>L</u> angua	ge: C++, thread	s 🗘	
Note: Preloading only works for programs linked against shared libraries. If your program is statically linked, you must relink it against the dmalloc library manually. Heap Debugging				
Fast	Balanced	Thorough	Custom	
Enabled Checks:	basic	More	Information	
Heap Overflow/Und	ges to detect out of bounds heat \mathbf{A} $$			
✓ Store stack <u>b</u> a	nsistency every 100 \bigstar he cktraces for memory allocations r these processes: 100% Select A		1%	
Help		ОК	Cancel	

Gotcha! Write OOB at res(k+2)



Memory Leak Detection

... and DDT in Offline Mode

arm

Confidential © 2018 Arm Limited

Three levels of heap debugging overhead



basic

•Detect invalid pointers passed to memory functions (e.g. malloc, free, ALLOCATE, DEALLOCATE,...)

check-fence

•Check the end of an allocation has not been overwritten when it is freed.

free-protect

• Protect freed memory (using hardware memory protection) so subsequent read/writes cause a fatal error.

Added goodness

 Memory usage, statistics, etc.

Balanced

•Overwrite the bytes of freed memory with a known value.

alloc-blank

free-blank

•Initialise the bytes of new allocations with a known value.

check-heap

•Check for heap corruption (e.g. due to writes to invalid memory addresses).

realloc-copy

 Always copy data to a new pointer when reallocating a memory allocation (e.g. due to realloc)

Thorough

•Check to see if space that was blanked when a pointer was allocated/freed has been overwritten.

check-blank

check-funcs

•Check the arguments of addition functions (mostly string operations) for invalid pointers.

See user-guide: Chapter 12.3.2

arm

Possible memory leak

Program is working great, but sometimes I run out of memory?

Exercise Outline

- Objectives
 - Use DDT in offline mode
 - Explore DDT's report logbook

Commands

- \$ make
- \$ ddt --offline \
 --output=report.html \
 aprun -n 4 \
 ./mmult3_f90.exe
- \$ xdg-open leak-report.html
- # Observe report

DDT in offline mode (--offline)

⊗					
<mark>johlin02@johlin02-VM:~/MUG18/01_walkthrough/3_memory_leak\$</mark> ddtofflineoutput=report.htmlmem- debug -n 4 ./mmult3_f90.exe Arm Forge 18.2.1 - Arm DDT					
MPI implementation : * number of processes : * number of nodes : Memory debugging enabled : * setting :	1				
0 : Size of the 0 : Initializing 1 : Receiving ma 2 : Receiving ma 3 : Receiving ma 0 : Sending matr 1 : Processing 2 : Processing 3 : Processing 1 : Sending resu 2 : Sending resu 0 : Receiving re 3 : Sending resu 0 : Writing resu 0 : Done.	g matrices atrices atrices ices 				

Offline log written to: '/home/johlin02/MUG18/01_walkthrough/3_memory_leak/report.htm

johlin02@johlin02-VM:~/MUG18/01_walkthrough/3_memory_leak\$ xdg-open report.html



View the memory leak report to see unfreed allocations

Allocations that are not freed when the program exits *could* be leaks

Click allocation to see function source



▼ allocate(mat_b(0:size*size-1))

- 59. if(myrank==0) then
- 60. allocate(mat_a(0:size*size-1))
- 61. allocate(mat_b(0:size*size-1))
- 62. allocate(mat_c(0:size*size-1))
- 63.
- 64. print *, myrank, ": Initializing matrices..."

Review source code to verify leak

← → C ① file:///home/johlin02/MUG18/01 walkthro	ugh/3_memory_leak/report.html#leaks 📩 🛧 👜 🔤 🤤 📿	M
		bookmar
Arm (Hat Higdon Iran) (Mait - John. Linic (Calendar - Johnan X Mission Control X Issue Navigator	DOOKINAI
Memory Leak Report		
This report shows unfreed memory allocations when the program fi where they were allocated.	nished executing. Clicking an item in the bar chart below will show additional details about the allocations	, including
All 4 ranks :		
	Legend	
Rank 0: 151.18 MB	mmult3 (mmult3.f90:62)	
Rank 1: 173.39 kB	Other	
Rank 2: 173.39 kB		
Rank 3: 173.39 kB		
Allocation data can also be <u>exported to CSV format</u> .	2)) on frank Ol-	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6	2)] on [rank 0]:	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	Source	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total)		
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	<pre>Source v allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then</pre>	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	<pre>Source v allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1))</pre>	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	Source ▼ allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1)) 62. allocate(mat_c(0:size*size-1)) 63.	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	<pre>Source v allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1)) 62. allocate(mat_c(0:size*size-1))</pre>	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function #0 mmult3 (mmult3.f90:62)	Source ▼ allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1)) 62. allocate(mat_c(0:size*size-1)) 63.	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function	Source v allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1)) 62. allocate(mat_c(0:size*size-1)) 63. 64. print *,myrank,": Initializing matrices*	
Allocation data can also be <u>exported to CSV format</u> . Largest allocation call path at [mmult3 (mmult3.f90:6 1 unfreed allocation (75.50 MB in total) Function #0 mmult3 (mmult3.f90:62)	Source v allocate(mat_b(0:size*size-1)) 59. if(myrank==0) then 60. allocate(mat_a(0:size*size-1)) 61. allocate(mat_b(0:size*size-1)) 62. allocate(mat_c(0:size*size-1)) 63. 64. print *,myrank,": Initializing matrices*	

DDT can also track leaks via the GUI

To see the equivalent of a leak report

- "Current Memory Usage" in the GUI shows all current, unfreed allocations
- To see something like the offline leak report, stop the program just before exit
 - Enable Control -> Default Breakpoints -> Exit
 - Run program to "exit"
 - Open "Current Memory Usage"

Also...

- "View pointer details" allows you to see where pointers were allocated, freed, and whether they point to a valid memory location
- Memory tracking also works for GPU allocations made with cudaMalloc

Another leak...

• Use either the GUI or a leak report to track down and fix the memory leak in the "memory-leak-mandel" exercise.

Profiling with MAP

...and Performance Reports



Confidential © 2018 Arm Limited

Profiling on Titan

Static binaries

- Need to link MAP libraries
- \$ module load forge/18.2.2
- \$ make-profile-libraries
- Generates libraries for your MPI and outputs instructions on how to link.

Dynamic binaries

- No need to link
- MAP will preload libraries into the binaries automatically
- We'll use this method today by adding dynamic to the link line

Improve performance

Efficient memory access

Confidential © 2018 Arm Limited



Fix inefficient memory access pattern

Revisiting the matrix multiply crash example

Exercise Outline

- Objectives
 - Discover Arm MAP's interface
 - Gather initial profiles of a MVAPICH2 application

Commands

\$ make

- \$ map --profile aprun -n 4 \
 ./mmult2_f90.exe
- \$ map mmult2_f90_4p*.map
- # Observe profile

Initial Result: SLOW

iohlin02@johlin02-VM: ~/MUG18/01_walkthrough/2_memory_accesses

johlin02@johlin02-VM:~/MUG18/01_walkthrough/2_memory_accesses\$ map --profile -n 4 ./mmult2_f90.exe Arm Forge 18.2.1 - Arm MAP

```
Profiling
                     : /home/johlin02/MUG18/01_walkthrough/2_memory_accesses/mmult2_f90.exe
Allinea sampler
                     : not preloading
MPI implementation : Auto-Detect (MVAPICH 2)
 number of processes : 4
  number of nodes
                   : 1
 Allinea MPI wrapper : not preloading
          1 : Receiving matrices...
          0 : Size of the matrices:
                                             1024 x
                                                           1024
          2 : Receiving matrices...
          3 : Receiving matrices...
           0 : Initializing matrices...
           0 : Sending matrices...
           1 : Processing...
           2 : Processing...
               Processing...
               Processing...
           2 : Sending result matrix...
           1 : Sending result matrix...
          3 : Sending result matrix...
          0 : Receiving result matrix...
          0 : Writing results...
          0 : Done.
MAP analysing program...
MAP gathering samples...
MAP generated /home/johlin02/MUG18/01 walkthrough/2 memory accesses/mmult2 f90 4p 1n 2018-08-05 23-0
2.map
```

johlin02@johlin02-VM:~/MUG18/01_walkthrough/2_memory_accesses\$

Initial profile

Find the hotspot: look for the line with the highest core time.



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] ...
    }
}</pre>
```

Spatial locality example

```
for (i=0 ; i < N*s; i+=s) {
    ... = ... x[i] ...
}</pre>
```



Memory Accesses and Cache Misses



arm

Answer: Transpose matrix and interchange loops

Transposing the matrix improves locality \rightarrow performance

Before

164	<pre>do i=0,size/nslices-1</pre>
165	do j=0,size-1
166	res=0.0
167	do k=0,size-1
168	res=A(i*size+k)*B(k*size+j)+res
169	end do
170	C(i*size+j)=res+C(i*size+j)
171	end do
172	end do

After

165	<pre>do i=0,size/nslices-1</pre>
166	do j=0,size-1
167	res=0.0
168	do k=0,size-1
169	res=A(i*size+k)*transB(j*size+k)+res
170	end do
171	C(i*size+j)=res+C(i*size+j)
172	end do
173	end do



Final profile

About 3x faster

Before



After





Debugging Imbalance MPI I/O

arm

Confidential © 2018 Arm Limited

Can we improve I/O performance?

R0 responsible for all file I/O after R1+ return results. Surely we can do better?

Exercise Outline

- Objectives
 - Use MAP's I/O profiling features
 - Use performance reports to quantify speedup
- Commands
 - \$ make
 - \$ map --profile aprun -n 4 \
 ./mmult5_f90.exe
 - \$ perf-report mmult5_f90_4p*.map
 - \$ xdg-open mmult5_f90_4p*.html

Performance report shows MPI bound



Summary: mmult5_f90.exe is MPI-bound in this configuration

Compute	46.7%		Time spent running application code. High values are usually good. This is low ; consider improving MPI or I/O performance first
MPI	53.0%		Time spent in MPI calls. High values are usually bad. This is high ; check the MPI breakdown for advice on reducing it
I/O	0.3%		Time spent in filesystem I/O. High values are usually bad. This is very low ; however single-process I/O may cause MPI wait times
This application	on run was	MPI-bound. A breakdow	n of this time and advice for investigating further is in the MPI section below.

Initial profile shows MPI_Finalize dominates

Time spent in MPI_Finalize is due to load imbalance in file I/O



Answer: improve scalability of I/O routines

Use MPI-IO to let all MPI ranks write their results to file simultaneously.

Before

97 if(myrank==0) then

```
100 do i=1,nproc-1
```

```
102 end do
```

```
103 else
```

107 end if

```
109 if(myrank==0) then
```

```
111 call mwrite(size, mat_c, filename)
```

113 endif

After

102	<pre>call MPI_FILE_OPEN(MPI_COMM_WORLD, & filename, & MPI_MODE_CREATE+MPI_MODE_WRONLY, & MPI_INF0_NULL, fh, ierr)</pre>
103	<pre>call MPI_FILE_SET_VIEW(fh, &</pre>
104	<pre>call MPI_FILE_WRITE_AT(fh, disp, mat_c, & slice, MPI_DOUBLE, st, ierr)</pre>
105	<pre>call MPI_BARRIER(MPI_COMM_WORLD, ierr)</pre>
106	<pre>call MPI_FILE_CLOSE(fh, ierr)</pre>

New approach: use MPI-IO for file output

Each MPI rank writes its results to it's own part of the output file

Before: runtime 13 seconds

	COIL
	Res
arm	Task
Performance	Mac
REPORTS	Star
	Tota

/home/johlin02/MUG18/01 walkthrough/5 MPI imbalance/mmult5 f90.exe 1 node (2 physical, 2 logical cores per node) 4 processes iohlin02-VM Mon Aug 6 2018 00:37:12 (UTC-04) 13 seconds /home/johlin02/MUG18/01 walkthrough/5 MPI imbalance



Summary: mmult5 f90.exe is MPI-bound in this configuration

Compute	46.7%	Time spent running application code. High values are usually good. This is low ; consider improving MPI or I/O performance first
MPI	53.0%	Time spent in MPI calls. High values are usually bad. This is high ; check the MPI breakdown for advice on reducing it
I/O	0.3%	Time spent in filesystem I/O. High values are usually bad. This is very low ; however single-process I/O may cause MPI wait times

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

After: runtime 5 seconds (2.6x speedup)

arm PERFORMANCE REPORTS

/home/johlin02/MUG18/01 walkthrough/5 MPL imbalance/solution/mmult6 f90.exempute 1 node (2 physical, 2 logical cores per node) 4 processes iohlin02-VM Mon Aug 6 2018 00:34:17 (UTC-04) 5 seconds /home/johlin02/MUG18/01 walkthrough/5 MPI imbalance/ solution



Summary: mmult6 f90.exe is Compute-bound in this configuration

Compute	74.5%	
MPI	20.1%	
I/O	5.3%	

Time spent running application code. High values are usually good. This is high; check the CPU performance section for advice

Time spent in MPI calls. High values are usually bad. This is low; this code may benefit from a higher process count

Time spent in filesystem I/O. High values are usually bad. This is low; check the I/O breakdown section for optimization advice

This application run was Compute-bound. A breakdown of this time and advice for investigating further is in the CPU section below. As little time is spent in MPI calls, this code may also benefit from running at larger scales.

Final profile shows balanced I/O and compute dominates

New approach is about 3x faster



GPU Debugging and Profiling With DDT and MAP

arm

Confidential © 2018 Arm Limited

GPU Debugging

- For many aspects, debugging on the GPU is very similar to debugging on the host
 - Adding breakpoints
 - Stepping through code
 - Inspecting variables, arrays, etc
 - Tracking memory
 - Memory error checking
- But there are important differences
 - Stepping will step the entire warp
 - Memory error checking is provided via cuda-memcheck

GPU Profiling

- Time spent waiting for accelerators
 - Determined by time spent in the CUDA (OpenACC, etc) API calls.
- GPU metrics include:
 - Percentage of time spent in global memory accesses
 - GPU temperature
 - Power consumption

• CUPTI data

- Which kernels were running and when
- On-GPU profile data

Thank You Danke Merci 谢谢 ありがとう Gracias **Kiitos** 감사합니다 धन्यवाद תודה

arm