An Analysis of Arm Graviton Systems Using Linaro Performance Reports

Beau.Paisley@linaro.org

Linaro 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 Linaro on Intel, AMD, Arm, IBM Power, Nvidia, AMD 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 Linaro

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

Linaro Forge

Debug
Linaro DDT

Profile
Linaro MAP

Analyse
Linaro
Performance Reports

Performance Engineering for any architecture, at any scale

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.

Bugs

- Correct application

Analyze before you optimize

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

I/O

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

Workloads

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

Communication

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

Memory

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

Cores

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

Vectorization

- Understand numerical intensity and vectorization level.
- Hot loops, unvectorized code and GPU performance reveleaed

Verification

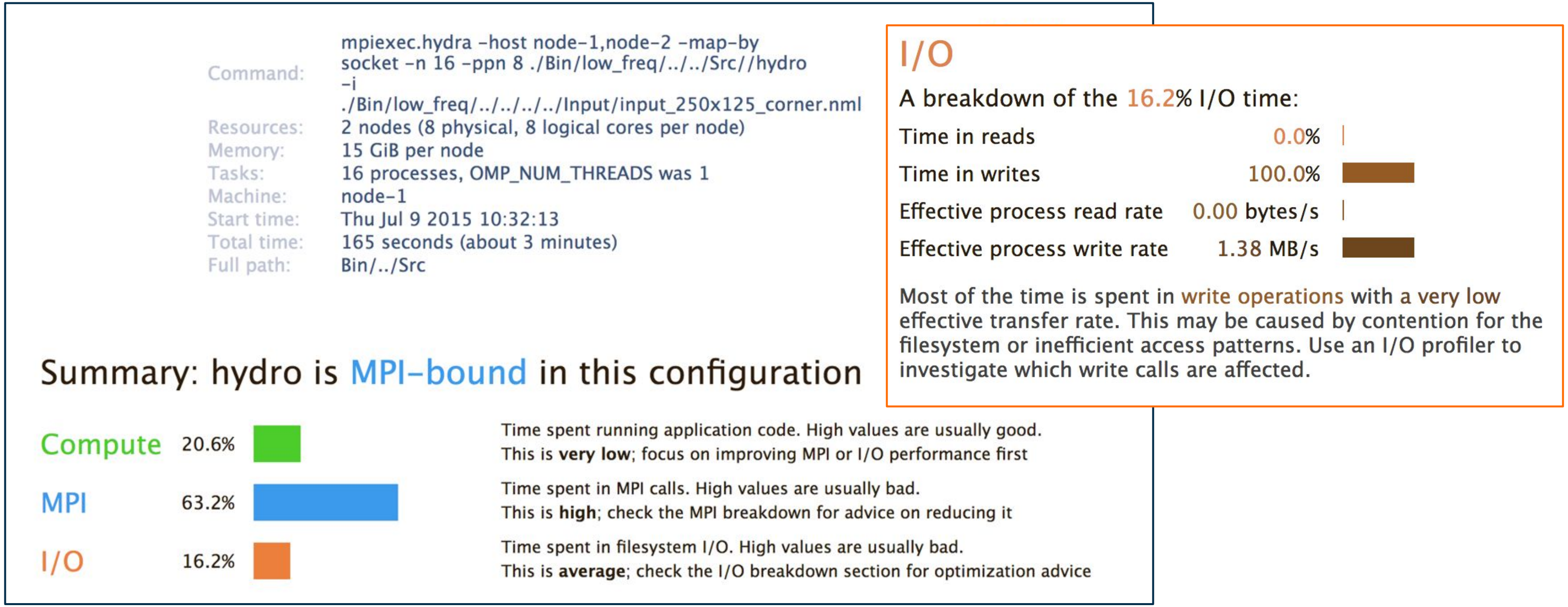
- Validate corrections and optimal performance

Key : Linaro Forge
Linaro Performance Reports

LinaroForge

Linaro Performance Reports

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

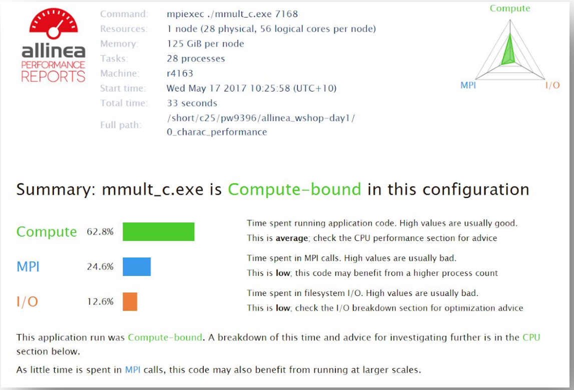


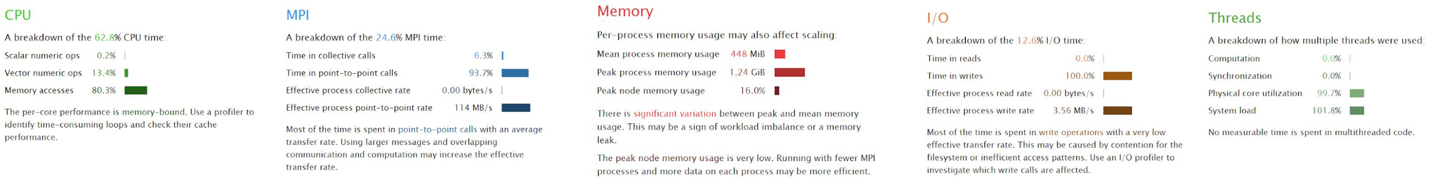
Understand application behaviour now

Set a reference for future work

- Choose a representative test cases with know results
- Analyse performance on existing hardware
- with Linaro Performance Reports
- Test scaling and note compiler flags

- Example
  $> perf-report mpirun –n 16 mmult.exe

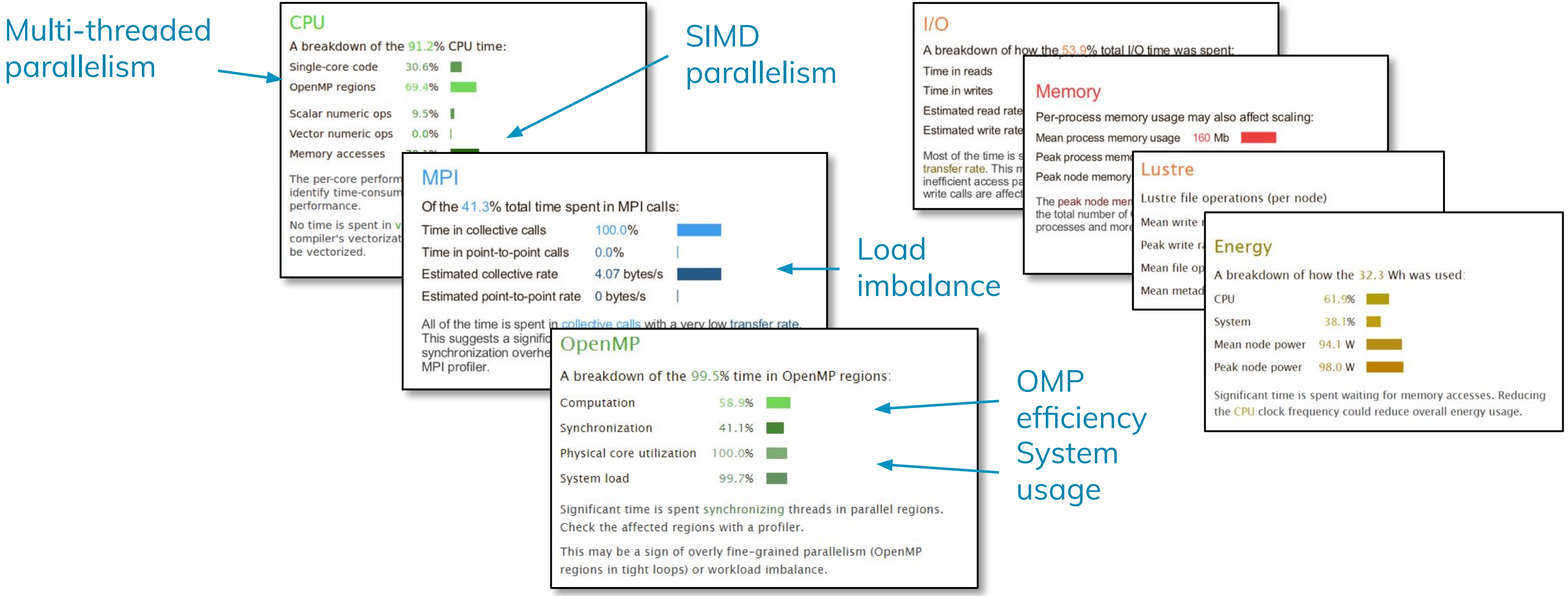




Linaro Forge

Linaro Performance Reports Metrics

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



Performance Reports command line options

```
$ perf-report --help
Arm Performance Reports 18.2.1 - Arm Performance Reports

Usage: perf-report [OPTION...] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] (mpirun|mpiexec|aprun|...) [MPI_ARGS] PROGRAM [PROGRAM_ARGS]
       perf-report [OPTION...] MAP_FILE

  --list-metrics                  Display metrics IDs which can be explicitly enabled or disabled.
  --disable-metrics=METRICS       Explicitly disable metrics specified by their metric IDs.
  --enable-metrics=METRICS        Explicitly enable metrics specified by their metric IDs.
  --mpiargs=ARGUMENTS             command line arguments to pass to mpirun
  --nodes=NUMNODES                configure the number of nodes for MPI jobs
  -o, --output=FILE               writes the Performance Report to FILE instead of an auto-generated name.
  -n, --np, --processes=NUMPROCS  specify the number of MPI processes
  --procs-per-node=PROCS          configure the number of processes per node for MPI jobs
  --select-ranks=RANKS            Select ranks to profile.
```

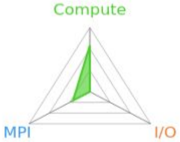
Linaro Forge

Fwd: [EXTERNAL] Re | wrf.exe - Performance R | wrf_armpl.exe - Perform | wrf_armpl_z.exe - Perfo | wrf_neoverse-512tvb.e | wrf_neoverse-512tvb_a | wrf_neoverse-512tvb_a

file:///home/beapai01/demo/WRFGraviton/wrf_8p_1n_8t_2022-10-14_13-39.html

arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main/wrf.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-82-58.ec2.internal
Start time:	Fri Oct 14 13:39:49 2022
Total time:	897 seconds (about 15 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf.exe is Compute-bound in this configuration

Compute	72.1%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	27.8%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	0.1%	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 Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU Metrics

Linux perf event metrics:

Single-core code	4.6%
OpenMP regions	95.4%
Cycles per instruction	0.98
L2D cache miss ratio	2.33
Stalled backend cycles	57.4%
Stalled frontend cycles	1.6%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 27.8% MPI time:

Time in collective calls	65.6%
Time in point-to-point calls	34.4%
Effective process collective rate	37.8 MB/s
Effective process point-to-point rate	669 MB/s

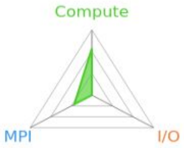
Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

Fwd: [EXTERNAL] Re | wrf.exe - Performance R | wrf_armpl.exe - Perform | wrf_armpl_z.exe - Perfo | wrf_neoverse-512tvb.e | wrf_neoverse-512tvb_a | wrf_neoverse-512tvb_a

file:///home/beapai01/demo/WRFGraviton/wrf_armpl_8p_1n_8t_2022-10-14_14-08.html

arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_armpl.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-82-58.ec2.internal
Start time:	Fri Oct 14 14:08:21 2022
Total time:	822 seconds (about 14 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_armpl.exe is Compute-bound in this configuration

Compute	71.5%	Time spent running application code. High values are usually good. This is **high**; check the CPU performance section for advice
MPI	28.4%	Time spent in MPI calls. High values are usually bad. This is **low**; this code may benefit from a higher process count
I/O	0.1%	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 Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

As little time is spent in MPI calls, this code may also benefit from running at larger scales.

CPU Metrics

Linux perf event metrics:

Single-core code	4.9%
OpenMP regions	95.1%
Cycles per instruction	1.03
L2D cache miss ratio	2.67
Stalled backend cycles	58.3%
Stalled frontend cycles	1.8%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 28.4% MPI time:

Time in collective calls	70.8%
Time in point-to-point calls	29.2%
Effective process collective rate	38.9 MB/s
Effective process point-to-point rate	838 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

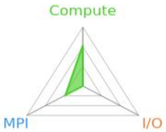
Linaro Forge

Sign in to your × | Inbox (33) - bxp × | My Drive - Goo × | WRFGraviton - × | wrf.exe - Performa × | wrf_armpl.exe - Pe × | wrf_armpl_z.exe - × | wrf_neoverse-512 × | wrf_neoverse-512 × | wrf_neoverse-512 ×

file:///home/beapai01/demo/WRFGraviton/wrf_neoverse-512tvb_8p_1n_8t_2022-10-16_11-07.html

arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:07:00 2022
Total time:	615 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_neoverse-512tvb.exe is Compute-bound in this configuration

Compute	68.8%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.0%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.1%	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 Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.2%
OpenMP regions	94.8%
Cycles per instruction	0.66
L2D cache miss ratio	1.13
Stalled backend cycles	65.3%
Stalled frontend cycles	2.2%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.0% MPI time:

Time in collective calls	68.0%
Time in point-to-point calls	32.0%
Effective process collective rate	45.1 MB/s
Effective process point-to-point rate	938 MB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

A breakdown of the 0.1% I/O time:

Time in reads	0.0%

OpenMP

A breakdown of the 94.8% time in OpenMP regions:

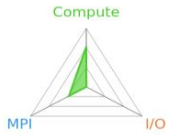
Computation	85.3%

Sign in to your | Inbox (33) - bxp | My Drive - Goo | WRFGraviton - | wrf.exe - Performa | wrf_armpl.exe - Pe | wrf_armpl_z.exe - | wrf_neoverse-512 | wrf_neoverse-512 | wrf_neoverse-512

file:///home/beapai01/demo/WRFGraviton/wrf_neoverse-512tvb_armpl_8p_1n_8t_2022-10-16_11-23.html

arm PERFORMANCE REPORTS

Command:	mpirun -n 8 --map-by socket:PE=8 ../main /wrf_neoverse-512tvb_armpl.exe
Resources:	1 node (64 physical, 64 logical cores per node)
Memory:	124 GiB per node
Tasks:	8 processes, OMP_NUM_THREADS was 8
Machine:	ip-172-31-25-35.us-west-2.compute.internal
Start time:	Sun Oct 16 11:23:15 2022
Total time:	570 seconds (about 10 minutes)
Full path:	/home/ec2-user/WRFV4.4/main



Summary: wrf_neoverse-512tvb_armpl.exe is Compute-bound in this configuration

Compute	68.7%	Time spent running application code. High values are usually good. This is **average**; check the CPU performance section for advice
MPI	31.1%	Time spent in MPI calls. High values are usually bad. This is **average**; check the MPI breakdown for advice on reducing it
I/O	0.3%	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 Compute-bound. A breakdown of this time and advice for investigating further is in the CPU Metrics section below.

CPU Metrics

Linux perf event metrics:

Single-core code	5.6%
OpenMP regions	94.4%
Cycles per instruction	0.78
L2D cache miss ratio	1.54
Stalled backend cycles	67.1%
Stalled frontend cycles	2.1%

A high number of cycles are stalled in the CPU. A high amount of memory accesses could be responsible for the non-exploitation of all the CPU cycles.

MPI

A breakdown of the 31.1% MPI time:

Time in collective calls	74.2%
Time in point-to-point calls	25.8%
Effective process collective rate	39.9 MB/s
Effective process point-to-point rate	1.26 GB/s

Most of the time is spent in collective calls with a low transfer rate. This can be caused by inefficient message sizes, such as many small messages, or by imbalanced workloads causing processes to wait.

I/O

A breakdown of the 0.3% I/O time:

Time in reads	0.0%

OpenMP

A breakdown of the 94.4% time in OpenMP regions:

Computation	84.9%

Thank you

~

www.linaroforge.com

Linaro Forge