BLIS User Level Profiler (DTL)

Dipal M Zambare

AMD

Agenda

Introduction to AOCL-BLIS

Debug and Trace Library

Q&A

Introduction to AOCL-BLIS

- AOCL (AMD optimizing CPU Libraries) are a set of numerical libraries tuned specifically for AMD EPYC™ processor family
- AOCL-BLIS is part of AMD optimized CPU Libraries
- AOCL-BLIS is a fork of BLIS library optimized for AMD processors.
- AOCL-BLIS resources:
  - Latest release of BLIS can downloaded from https://developer.amd.com/amd-aocl/
  - Source code for BLIS is available on GitHub https://github.com/amd/blis
  - Technical support is available via https://developer.amd.com/amd-optimizing-cc-compiler-aocc-technical-support/

Contents

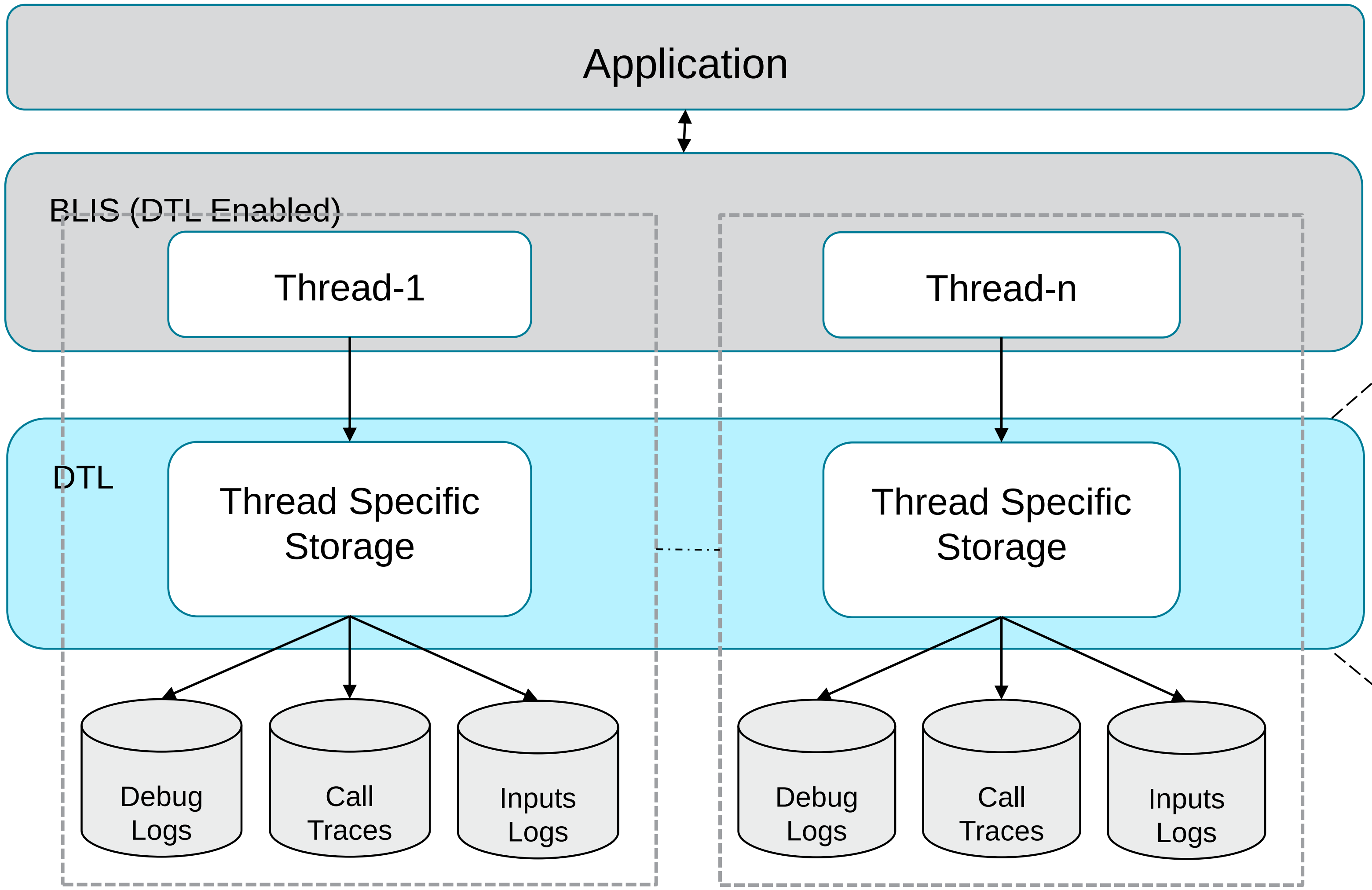
What is DTL (Debug and Trace Library)?

- DTL is primarily a thread safe logging feature
- DTL logs provides a mechanism for the end user to:
  - Understand program flow
  - Application profiling
  - Timing (hotspot) analyses
  - Debugging
  - Crash analyses
- DTL logs are stored in plain text files, no specific tools, profilers are needed with DTL
- DTL is implemented as a static library
- DTL is integrated in AOCL-BLIS and AOCL-libFLAME

DTL Features

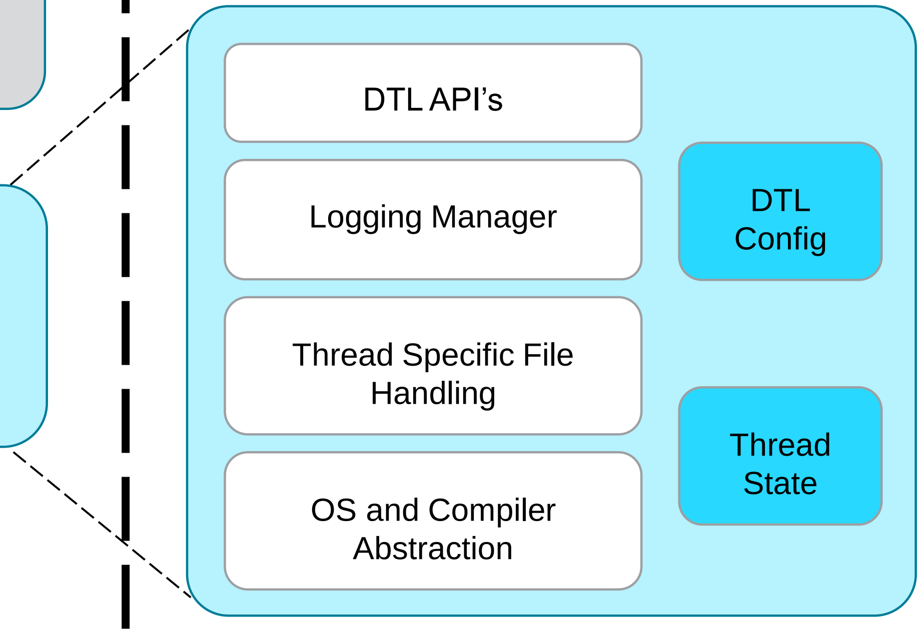
- Logging of input values, All input logs are printed as per BLAS API’s order
- Logging of thread, timing information
- Function call traces
- Logging of user defined messages, data structures
- Input logs & timing information, Traces and Data dump can be enabled separately
- Thread Safe Logging
- Compile time enable/disablement
- OS and Compiler independent implementation
- DTL is implemented as a library
- Zero impact on performance when DTL is disabled (the DTL code is excluded at the compile time)

DTL Architecture



System Overview

DTL is library for logging traces, debug and profiling information in a thread safe manner.



DTL Implementation

DTL – Thread Safe Logging

- Thread safe logging suitable for
  - Single threaded
  - Multi threaded
  - Multi instance
  - Multi node
- Thread safety is achieved by separating logs for each thread in a separate file.
- File name determine the unique source of the logs.

File names for input and debug logs.

```
P<Process id>_T<Thread id>_aocldtl_log.txt

e.g. P3504033_T0_aocldtl_log.txt
```

File names for traces.

```
P<Process id>_T<Thread id>_aocldtl_trace.txt

e.g. P3504033_T4_aocldtl_trace.txt
```

Challenges in Application profiling

- Different applications use BLIS differently, understanding the application profile is important for application specific, highly optimized solution.
- BLIS has around 40 API
- Each API has multiple variants:
  - For example, TRSM (Triangular Solve Matrix Equations)

  dtrsm(side, uplo, transa, diag, m, n, alpha, a, lda, b, ldb)

Side	Uplo	Transa	Diag
Left or Right	L (Lower) or U (upper)	T – Transpose or no transpose	Unit or non-unit diagonal

  - There will be around 16 variants of TRSM and the benchmarks performance can depend on any of these variants.
- Performance also depends on dimensions (m, n, lda and ldb) of the matrices. Based on dimensions we need different algorithm for a given variant.

Challenges in Application profiling

- Traditionally we use system level profiles like uprof.
- These reports can give a general idea, but it is not enough.

```
 54.52%  xhpl             libblis-mt.so              [.] bli_dgemmsup_rv_haswell_asm_6x8m
 13.86%  xhpl             libgomp.so.1.0.0           [.] 0x000000000001fb56
  6.32%  xhpl             [unknown]                  [k] 0xffffffffbb10df3c
  3.31%  xhpl             mca_btl_vader.so           [.] mca_btl_vader_component_progress
  1.87%  xhpl             xhpl                       [.] HPL_rand
  1.67%  xhpl             xhpl                       [.] HPL_lmul
  1.46%  xhpl             libopen-pal.so.40          [.] opal_progress
  1.22%  xhpl             xhpl                       [.] HPL_dlaswp10N
```

- For example, the dgemmsup kernels can be invoked from GEMM as well as TRSM code paths.
- The DTL logs identify critical API’s and inputs that contributes most to the performance.

DTL Usage - Application Profile

Example Profiling logs:

```
dgemm_ D N T 4536 108 0 -1.000000 0.000000 4536 216 1.000000 0.000000 4536 nt=4 0.058 ms 0.000 GFLOPS
dgemm_ D N T 4536 56 0 -1.000000 0.000000 4536 216 1.000000 0.000000 4536 nt=-4 0.003 ms 0.000 GFLOPS
idamax_ D 4536 1
dcopy_ D 216 4536 1
dger_ d 4535 2 -1.000000 0.000000 1 1 4536
dcopy_ D 216 4536 1
dcopy_ D 216 4536 1
dscal_ D 1.526621 0.000000 4534 1
daxpy_ D 4534 -0.466632 0.000000 1 1
idamax_ D 4534 1
dger_ d 4534 1 -1.000000 0.000000 1 1 4536
dcopy_ D 216 4536 1
dcopy_ D 220 1 1
dscal_ D -1.019477 0.000000 4532 1
dtrsm_ d R U N U 4 4 216 216 1.000000 0.000000
dgemm_ D N T 4532 4 4 -1.000000 0.000000 4536 216 1.000000 0.000000 4536 nt=4 0.058 ms 2.500 GFLOPS
```

- Profile logs can help identify critical API’s and input sizes.
  - What API’s are called?
  - At what frequency?
  - With what inputs?

DTL Usage – Timing/Performance Analyses

- The DTL logs for selected Level 3 APIs include performance information such as
  - Input values
  - Number of threads used
  - Time taken by the API
  - FLOPS achieved

```
$BLIS_NUM_THREADS=1 ./test_gemm_blis.x input.txt output.txt
BLIS Library version is : AOCL BLIS 3.1
~~~~~~~~~~_BLAS   m        n        k        cs_a     cs_b     cs_c     gflops
data_gemm_aocl  1000     2000     3000     4000     5000     6000     13.481, 0.890153

$ls P*.txt
P183294_T0_aocldtl_log.txt  P183294_T0_aocldtl_trace.txt

$cat P183294_T0_aocldtl_log.txt
dgemm_ D N N 1000 3000 2000 0.900000 0.000000 4000 5000 -1.100000 0.000000 6000 nt=1 890.131 ms 13.481 GFLOPS
```

DTL Usage - Traces

Function call tracing with configurable nested levels

```
Logging limited to 5 nested levels

In bli_gemm()...
In bli_gemmsup()...
        In bli_gemmsup_ref()...
                In bli_gemmsup_int()...
                        In bli_gemmsup_ref_var2m()...
                        In bli_spackm_sup_b()...
                        << output snipped>>
                        Out of bli_gemmsup_ref_var2m()
                Out of bli_gemmsup_int()
        Out of bli_gemmsup_ref()
Out of bli_gemm()


Logging Limited to 1 nested level


In bli_gemm()...
Out of bli_gemm()
```

- DTL supports call traces for up to 9 nested levels.
- The logging levels are configured at compile time.
- Optional time stamping for each trace.

DTL Usage – Crash Identification

- DTL logs can help in debugging a crash by identifying:
  - Input which caused crash.
  - Last function called before the crash.
- Function Name Identification
  - Each function logs entry (In) and exit (Out) traces.
  - The crashed function will have the entry trace but not the exit trace.
- Inputs Identification (Supported For selected API's)
  - The input logs have two parts i.e., the inputs values and the stats.
  - The inputs are available when the API is invoked.
  - Stats are available only if the API has completed successfully.
  - If the API with given input has crashed, only input values are printed (stats will be not printed).
- Please check the example on the next slide.

DTL Usage – Crash Identification

```
$ BLIS_NUM_THREADS=1 ./test_gemm_blis.x input.txt output.txt
~~~~~~~~~~_BLAS  m       n       k       cs_a    cs_b    cs_c    gflops
data_gemm_aocl  1000     2000    3000    4000    5000    6000    13.481, 0.890153
Segmentation fault (core dumped) -> Crash

$ ls P*.txt
P183294_T0_aocldtl_log.txt  P183294_T0_aocldtl_trace.txt    -> DTL output files

$ cat P183294_T0_aocldtl_log.txt
dgemm_ D N N 1000 3000 2000 0.900000 0.000000 4000 5000 -1.100000 0.000000 6000 nt=1 890.131 ms 13.481 GFLOPS
dgemm_ D N N 100 100 100 0.900000 0.000000 104 104 -1.100000 0.000000 104 -> Inputs which caused the crash

$ cat P183294_T0_aocldtl_trace.txt
           In bli_gemm_packa()...
                      In bli_gemm_int()...
                                 In bli_gemm_ker_var2()...
                                 In bli_dgemm_ker_var2()...
                                            In bli_dgemm_haswell_asm_6x8()...->Crash location (Function Name)
```

DTL Limitation

- The input and trace logs needs to added manually in each function of interest.
- Runtime control is not available to minimize performance impact.
- Timing logging is supported only for selected level – 3 APIs.
- Based on input dataset, DTL logs may take huge disk space.
- Performance will degrade when DTL traces are enabled.

questions?

COPYRIGHT AND DISCLAIMER

AMD