AOCL BLIS framework changes

Edward Smyth

AMD

Agenda

Introduction

- AOCL – AMD optimizing CPU Libraries
  - AOCL are a set of numerical libraries tuned specifically for AMD EPYC™ processor family
  - BLIS is part of this AOCL
  - https://developer.amd.com/amd-aocl/
  - Latest source code for BLIS is available on GitHub https://github.com/amd/blis

- For any issues or queries regarding the libraries, please contact toolchainsupport@amd.com

- Latest version is AOCL 4.1
  - Includes further AVX512 optimizations for “Zen 4”
  - Enhanced LPGEMM functionality
  - Framework enhancements for OpenMP and dynamic dispatch

Enhancements to code path selection at runtime

BLIS code path selection at runtime

- BLIS supports configuration families to enable multiple microarchitecture sub-configurations within a single library build
- Used in AOCL to support different optimized code paths for different Zen processor generations
  - Enabled using the “amdzen” build configuration
- Sub-configuration chosen automatically at runtime based on hardware characteristics
- Desirable to be able to select a sub-configuration different from hardware’s default choice
  - To ensure consistent results across different generations of hardware
  - Useful for testing
- Controlled using BLIS_ARCH_TYPE environment variable
  - Set BLIS_ARCH_DEBUG to print information on sub-configuration selected

BLIS code path selection at runtime

- BLIS_ARCH_TYPE mechanism uses an integer value corresponding to the enumeration value in the internal datatype for all sub-configurations across all supported architectures
  - e.g., AOCL 4.0, BLIS_ARCH_TYPE=6 gave Zen4 code path
- Problems
  - Difficult to remember correct value
  - Value may change as new sub-configurations are added
    - e.g., AOCL 3.2, BLIS_ARCH_TYPE=6 gave Zen3 code path
  - Invalid choice results in BLIS_ARCH_TYPE=0, which just happens to be “BLIS_ARCH_SKX”
    - Error if this is not enabled in configuration family
    - Even if using x86-64 processor, invalid instruction error if AVX512 is not supported
  - Possibility of runtime error from a “stray” BLIS_ARCH_TYPE env var setting

AOCL BLIS enhancements to code path selection at runtime

- BLIS_ARCH_TYPE now supports meaningful names as well as enumeration value
  - e.g., BLIS_ARCH_TYPE=zen3 or BLIS_ARCH_TYPE=generic
  - Name is case-insensitive
- Invalid choice still results in BLIS_ARCH_TYPE=0, but this is now always an error
- To avoid the “stray” BLIS_ARCH_TYPE setting problem
  - Configure option to rename BLIS_ARCH_TYPE env var to another name
  - Configure option to disable BLIS_ARCH_TYPE functionality
    - Automatic choice of code path would continue but couldn’t be overridden
- Added BLIS_MODEL_TYPE as a sub-sub-configuration within a given BLIS_ARCH_TYPE
  - Doesn’t duplicate code or object files
  - Incorrect BLIS_MODEL_TYPE results in default config for that value of BLIS_ARCH_TYPE

Improvements to OpenMP support

Parallelism options in BLIS

- Library can be compiled to be serial or use OpenMP or pthreads parallelism
  - HPX parallelism is not currently implemented in AOCL BLIS
  - Not covering pthreads in this talk
- In OpenMP parallel builds, hierarchy of mechanisms for controlling threading
  1. BLIS locally, manual, using BLIS expert APIs
  2. BLIS locally, automatic, using BLIS expert APIs
  3. BLIS globally, manual, API > env variable, using BLAS or BLIS APIs
  4. BLIS globally, automatic, API > env variable, using BLAS or BLIS APIs
  5. OpenMP, API > env variable, using BLAS or BLIS APIs
- Where
  - Manual = user specifies parallelism to use for each loop in blocked GEMM operation
  - Automatic = set single number of threads, BLIS chooses how to divide these among loops
- (See https://github.com/amd/blis/blob/master/docs/Multithreading.md for more info)

How is parallelism implemented in BLIS?

- BLIS uses global data structure to manage threading information (*global_rntm*)
  - Passes data between internal BLIS functions within a single BLIS call
  - Maintains state between subsequent BLIS calls
  - Maintains global settings for BLIS routines called within higher level parallelism
- Function makes local copy of *global_rntm*, may alter settings in local copy e.g., due to AOCL_DYNAMIC functionality
- Global data structure imposes limitations on how BLIS can be used, with respect to OpenMP nested parallelism and changing numbers of threads between BLIS calls
- AOCL team have made improvements to OpenMP parallelism in AOCL BLIS over several releases

What are the OpenMP issues?

- OpenMP parallelism settings can be changed within user program
    - Changing thread count from one API call to the next via OpenMP API calls
    - Nested parallelism
- Need to check OpenMP ICVs at every BLIS API call
    - Considering value from *omp_get_max_threads()* and OpenMP active levels and current level
- With nested parallelism, different user threads may set different OpenMP values
    - BLIS threading information needs to be specific to each calling user thread
- If OpenMP nested parallelism occurs, but level within BLIS is inactive
    - Threads will not be created inside BLIS APIs
    - This is irrespective of BLIS-specific threading settings
    - Need to ensure we account for this in setting thread counts inside BLIS

Improvements to OpenMP support

- Keep *global_rntm*
  - Initialize once from BLIS environment variables
  - Update from any calls to *bli_thread_set_num_threads()* or *bli_thread_set_ways()*

- Also have rntm specific to each user thread (*tl_rntm*)
  - Using TLS (e.g., C *__thread* specifier)
  - Used just for threading info
  - Other parts of rntm remain in *global_rntm*

- Update *tl_rntm* at start of every relevant BLIS routine
  - Check for any changes in *global_rntm*
  - Update threading info (as needed) from OpenMP ICVs

- BLIS-specific ways of setting threading work as before
  - But takes account of OpenMP active levels

questions?

DISCLAIMER AND ATTRIBUTIONS

AMD