December 11, 2021

Homogeneous and Heterogeneous Implementations of a tridiagonal solver on Intel® Xeon® E-2176G with oneMKL getrs() routine

By

Oluwatosin S. Oluseyi

1.1. Characteristics of Development Environment and Processor used

Development Environment
Environment: Intel DevCloud **Accessed from:** HP-15-R029WM, Intel® Pentium® CPU N3540 @2.16GHz, 4 Core(s), 4 Logical Processor(s) **File Storage:** 220GB **RAM:** 192GB **Terminal Interface:** Linux
Characteristics of Processor
Processor: Intel® Xeon® E-2176G **Frequency**: 3.70 GHz (4.70 GHz turbo) **Cache:** 12 MB Intel® Smart Cache **Cores:** 6 (12 Logical) **Threads:** 12 **Max Memory Size:** 128GB **Memory Type:** DDR4-2666 **Max Memory Bandwidth:** 41.6 GB/s **Processor Graphics:** Intel® UHD Graphics P630 [0x3e96] **Frequency:** 0.35 – 1.20 GHz **Execution Units:** 24 (For more information, see: https://www.intel.com/content/www/us/en/products/sku/134860/intel-xeon-e2176gprocessor-12m-cache-up-to-4-70-ghz/specifications.html)

1.2. What is getrs() : (https://oneapi-src.github.io/oneMKL/domains/lapack/getrs.html#onemkl-lapack-getrs)

The getrs() routine belongs to the oneapi::mkl::lapack namespace, it has a unified shared memory (USM) and Buffer version. It solves a system of linear equations with an LU-factored square coefficient matrix, with multiple right-hand sides. The getrs() routine supports float, double, std::complex and std::complex precisions and can be executed on Host, CPU and GPU devices.

Since the getrs() routine works with an LU-factored square coefficient matrix the getrf() routine must be called first to compute the LU-factorization of the matrix as A = P*L*U, where P is a permutation matrix, L is the lower triangular matrix with unit diagonal elements (lower trapezoidal if m > n for an m*n matrix) and U is the upper triangular matrix(upper trapezoidal if m < n for an m*n matrix). This routine uses partial pivoting with row interchanges, it has a USM and Buffer version and can be executed on Host, CPU and GPU devices like the getrs() routine.

1.3. What is a Tridiagonal System ?

Tridiagonal linear systems are a special type of banded linear system having non-zero elements on the main-diagonal, sub-diagonal and super-diagonal; they have specific applications in fluid dynamics, computer graphics, modeling of medical problems, finance, and many more.

1.4. Implementations

Homogeneous implementation: Implementation is executed on either host or gpu device.
Heterogeneous implementation: Queues are used to specify what section of code uses a particular device.

1. getrs_usm.cpp: Homogeneous implementation based on Unified Shared Memory version of getrs(). This implementation can be executed on either host or gpu device.

2. getrs_buffer.cpp: Homogeneous implementation based on buffer version of getrs implementation

3. getrs_usm_het.cpp: Heterogeneous implementation based on Unified Shared Memory version of getrs(). Queues are used to define that:
1. The factorization of the matrix is done on the GPU device;
2. The solution of the matrix is computed on the Host device (in other words, do the getrf() on the GPU and the getrs() on the CPU).

For simplicity, I will call this USM Heterogeneous Version 1.

4. getrs_usm_het2.cpp: Heterogeneous implementation based on Unified Shared Memory version of getrs(). Queues are used to define that:
1. Memory for the matrix as well as the pivot array is allocated on the GPU device while memory for the right-hand side array is allocated on the Host device;
2. The pointer to the scratchpad memory which is used for storing intermediate results as well as the size of the scratchpad memory for getrf() is allocated on the GPU device while that of getrs() is allocated on the Host device;
3. Both getrf and getrs are performed on the Host device.

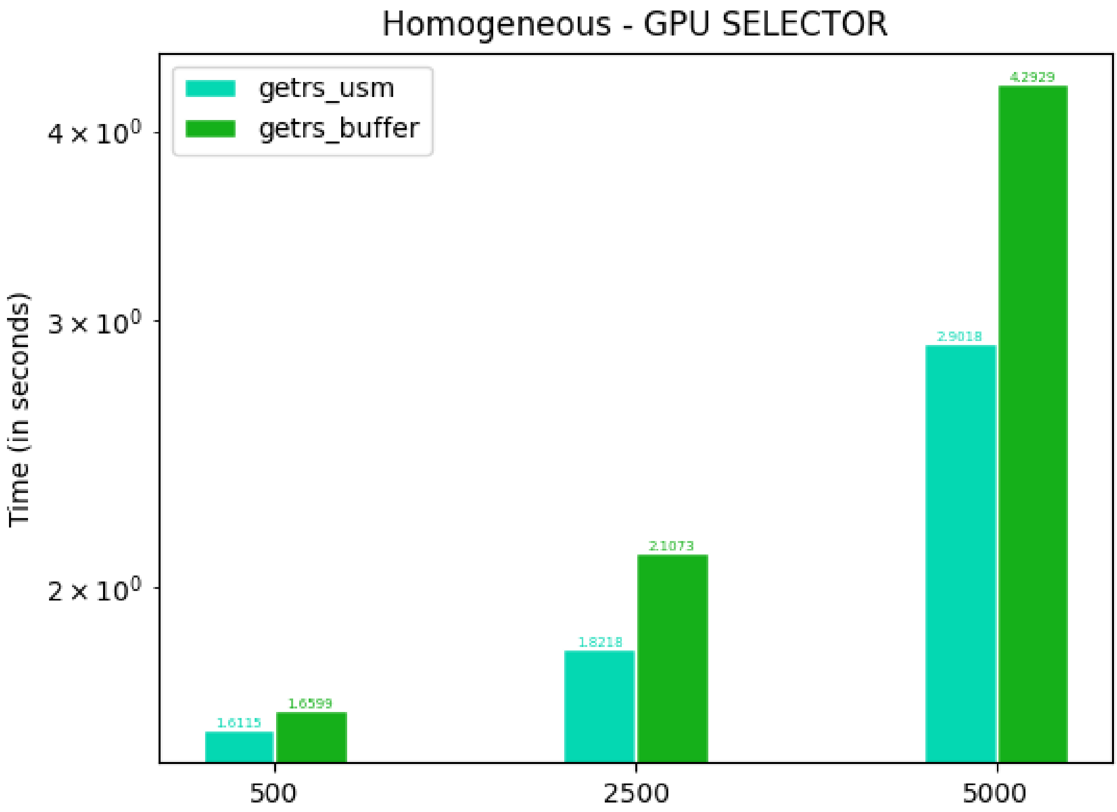
For simplicity, I will call this USM Heterogeneous Version 2.

5. **getrs_buffer_het.cpp:** Heterogeneous implementation based on buffer version of getrs(). Queues are used to define that:
1. The factorization of the matrix is done on the GPU device;
2. The solution of the matrix is computed on the Host device (in other words, do the getrf() on the GPU and the getrs() on the CPU).

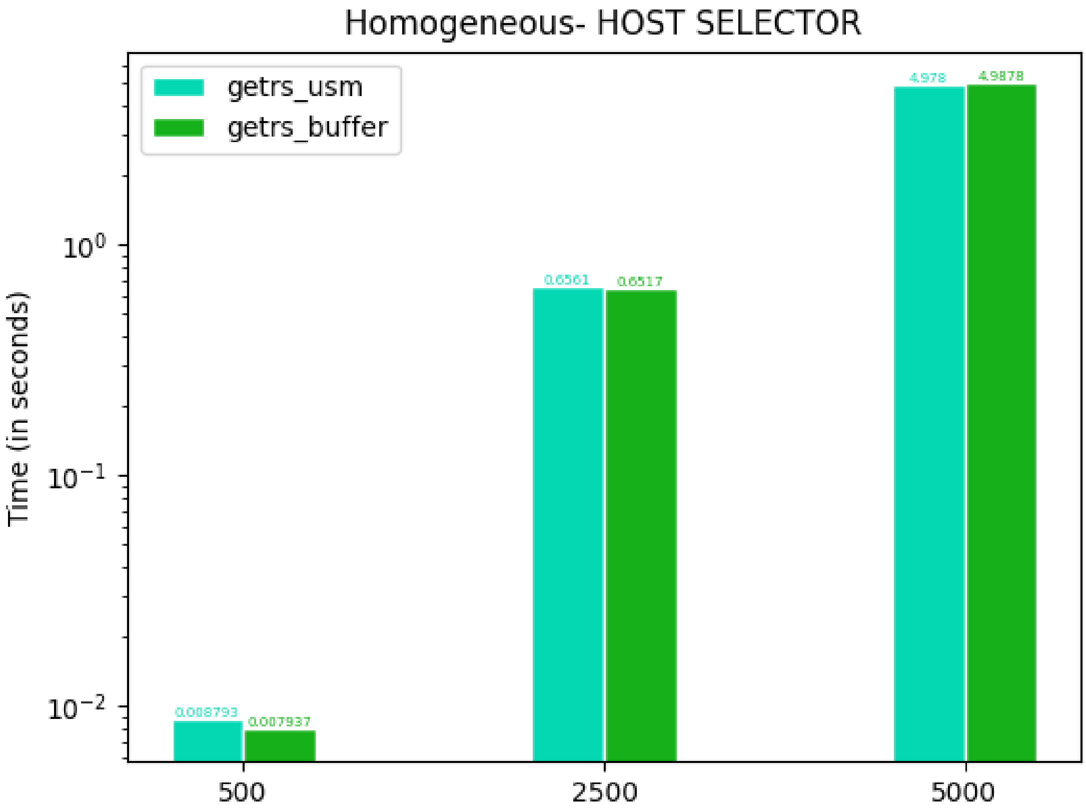
For simplicity, I will call this BUFFER Heterogeneous Version 1.

1.5. Results

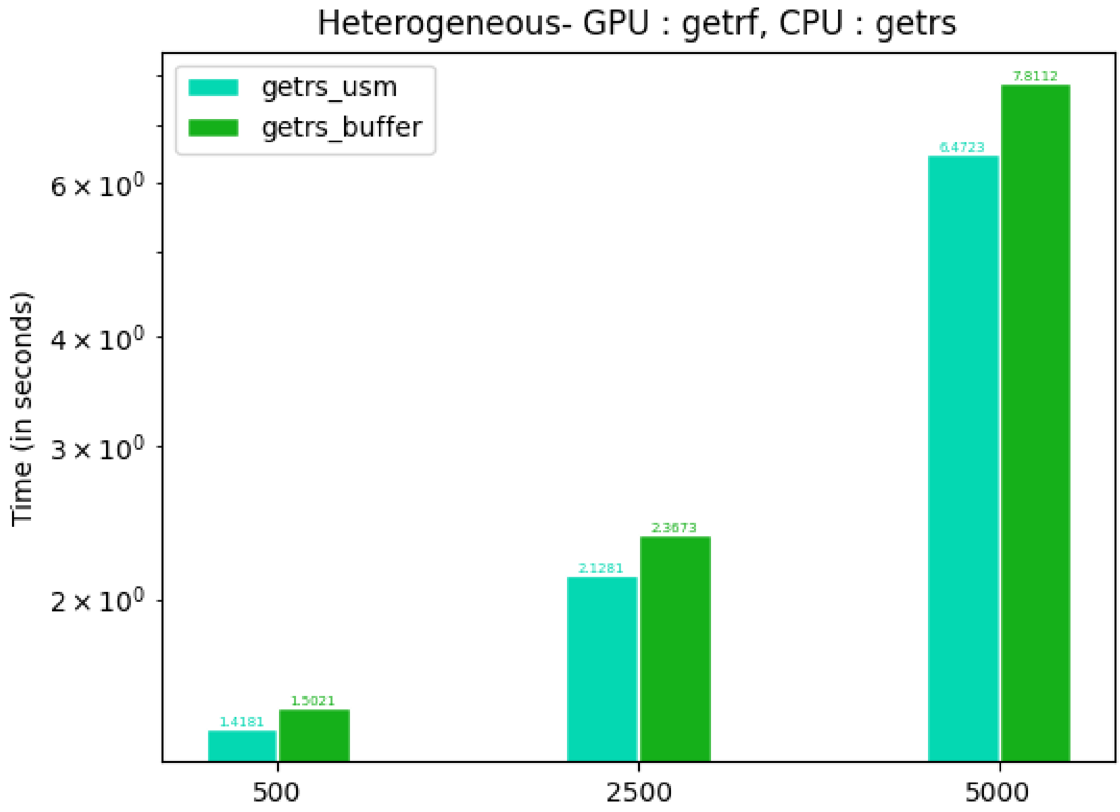
Test sizes = 500, 2500, 5000.



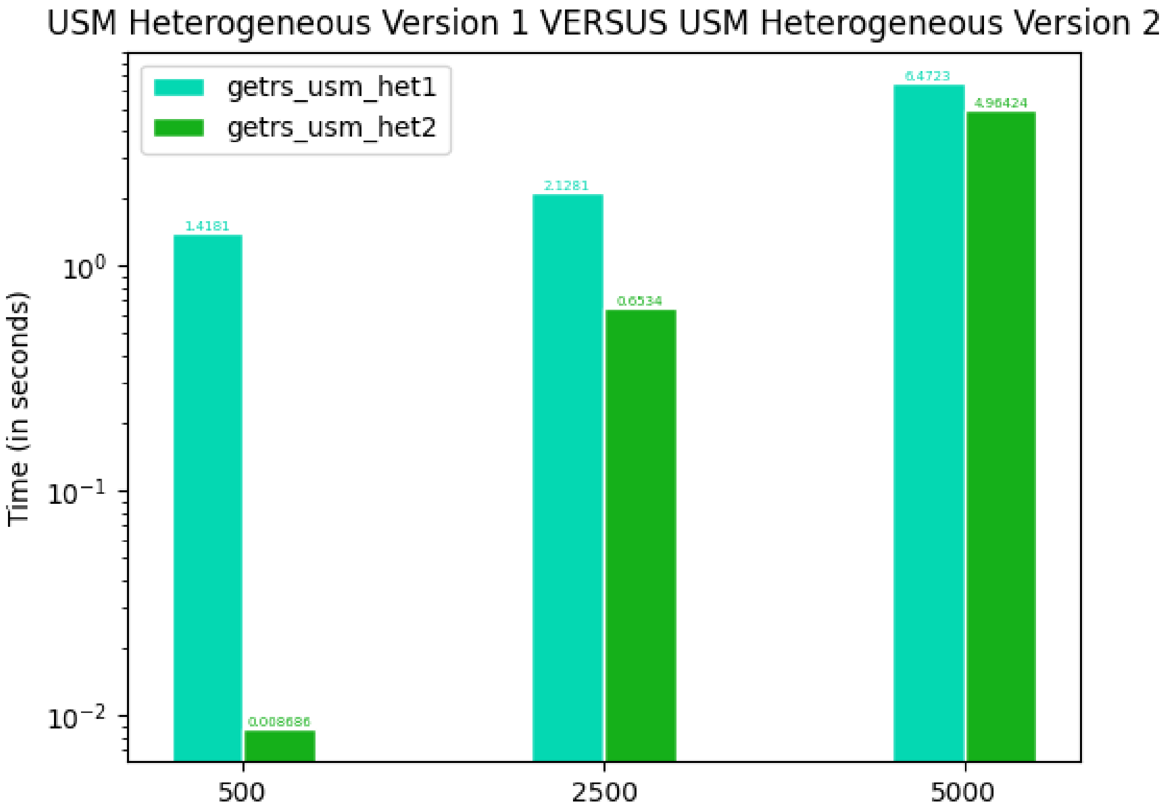
Comment: The figure above is results for Homogeneous getrs USM and BUFFER implementations executed on gpu device. We see that for all test sizes, getrs() usm version executes faster.



Comment: The figure above is results for Homogeneous getrs USM and BUFFER implementations executed on host device. We see that for test sizes, 500 and 2500, getrs() buffer version executes faster, however, for test size, 5000, getrs() USM, executes faster than getrs() buffer version (the difference is not so much though).



Comment: The figure above is results for Heterogeneous getrs USM and BUFFER implementations executed on host and gpu devices. We see that for all test sizes, the USM version executes faster.



Comment: The figure above is results for Heterogeneous getrs USM implementations (version 1 and 2) executed on host and gpu devices. We see that for all test sizes, the USM Heterogeneous Version 2 executes faster.

Final comment: Best performing implementation is USM Heterogeneous Version 2.