COEN 193 Report Fall

Anthony Bryson

December 2023

1 Introduction

The objective of this research is to analyze hardware events resulting from the execution of target parallel applications. Machine learning will be applied to perform analysis on these events to determine their affect on the performance of said applications.

For this quarter my main goals were to familiarize myself with the LAPACK [1], OpenBLAS [2], and Intel MKL [3] libraries for linear algebra computation, run applications with utilizing these libraries on the WAVE HPC [4] and time their execution, and use PAPI [5] for collecting performance counter data on the applications. My focus was on testing the LAPACK procedures for *dgels*, *dgemm*, and *dgeqrf*. Specifically, I looked at these applications when compiled with the OpenBLAS library versus the Intel MKL library.

2 Progress

My progress for the quarter can be described in three main sections. The first few weeks were spent performing the setup to get my code to run on the WAVE system. Once this was done I turned to creating code to test the execution time of the various procedures. My last few weeks were focused on getting performance counting data from the programs I created earlier.

2.1 Setup

The setup process involved installing the necessary libraries to the HPC and writing simple example programs for the procedures. It took more time than anticipated to make these libraries work, but I managed to get them setup once I became familiar with the modules available with the WAVE system. After a little more time spent figuring out which library files I needed to link and include to compile some test programs, I had everything working with both OpenBLAS and Intel MKL.

2.2 Measuring Execution Time

After getting past the setup obstacles, I was able to start writing and running test applications for each of the *dgels*, *dgemm*, and *dgeqrf* procedures. I compiled versions for each that utilized OpenBLAS and versions that used Intel MKL, then ran them on the HPC to measure the execution times of the procedures.

I ran the test applications several times, altering the size of the matrices and the number of threads with each execution. The resulting execution times were stored in a csv file which I later used to create visuals for the data. I did so using Python by writing a few scripts to organize and plot the data on to graphs, which allowed me to see the relationship between the different libraries and number of threads used for each problem size.

2.3 Measuring Performance Counters

Once these graphs were created, I was introduced to PAPI for gathering performance counter information. Using this library did not require significant changes to how I was running the test applications, so I decided to modify them with the necessary PAPI function calls to gather the performance counter data. For my initial attempt at this, I gathered data on the total clock cycles and total instructions executed. I re-ran the test applications with the same sized matrices and number of threads, and then graphed these alongside the execution times to see the relation.

3 Results

For each of the *dgels*, *dgemm*, and *dgeqrf* procedures, I gathered csv files of their execution times and of their clock cycles and total instructions. Additionally, I graphed a roughly representative selection of this data using Python.

3.1 Execution Times

The data on the execution times of the procedures was output into the csv files as shown in Figure 1, separated by which BLAS library was utilized.

```
time,m,n,k,threads,
5.22923,10000,10000,10000,4,
1.88465,10000,10000,10000,12,
1.19365,10000,10000,10000,24,
1.16625,10000,10000,10000,48,
1.17192,10000,10000,10000,72,
1.17155,10000,10000,10000,96,
15.3814,10000,10000,30000,4,
5.4346,10000,10000,30000,12,
3.14448,10000,10000,30000,24,
2.63261,10000,10000,30000,48,
2.62814,10000,10000,30000,72,
2.55992,10000,10000,30000,96,
```

Figure 1: DGEMM data of execution times

I then selected a few data points, organized by matrix size, to graph to visualize the scaling of the procedures. To avoid cluttering the graphs with too many lines, I separated them into large and small matrices of interest, with Figure 2 showing some of the smaller matrices and Figure 3 showing a selection of the larger ones.

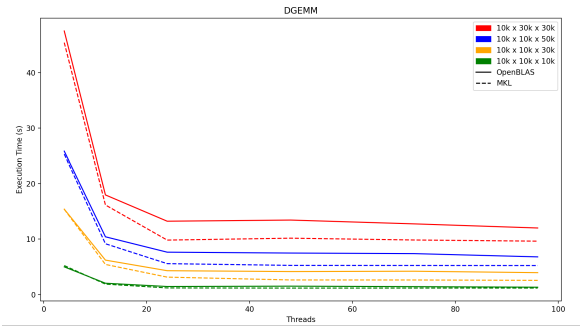

Figure 2: DGEMM execution times of select small matrices

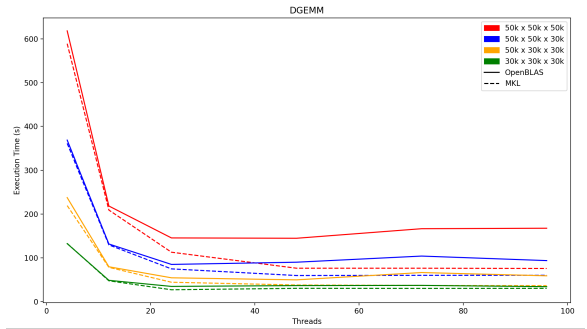

Figure 3: DGEMM execution times of select large matrices

3.2 Performance Counters

The data on the clock cycles and total instructions counters were organized in the same manner. Figure 4 shows a snippet of the raw data in the csv file.

```
cycles,instructions,m,n,k,threads,
16659465108,55802162964,10000,10000,10000,4,
5913746336,18760701439,10000,10000,10000,12,
3880531093,9532591892,10000,10000,10000,24,
3280982341,4878246590,10000,10000,10000,48,
3156880492,3229871559,10000,10000,10000,72,
2979753059,2560881960,10000,10000,10000,96,
50713877026,167342714461,10000,10000,30000,4,
17622189197,56216824738,10000,10000,30000,12,
9837256638,28425711725,10000,10000,30000,24,
8657528344,14530813718,10000,10000,30000,48,
8975838710,9679539917,10000,10000,30000,72,
8823702097,7489063344,10000,10000,30000,96,
```

Figure 4: DGEMM data of performance counters

As an initial visualization, I graphed these counters alongside the execution times of the corresponding matrices to see their changes relative to each other. Figure 5 demonstrates how these values are graphed for each of a number of select matrices.

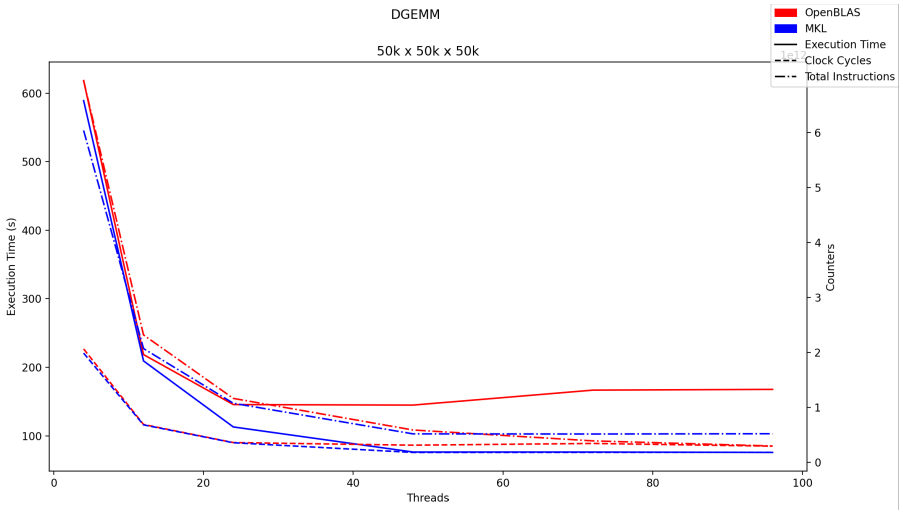

Figure 5: DGEMM execution and performance counters of select matrix

4 Future Work

Starting next quarter, I will focus on polishing up my existing test applications and correcting a few mistakes I have made. Most notably is the method with which I am measuring performance counters with PAPI. However I also need to rethink how I gather my data on execution time when in conjunction with the performance counters.

4.1 Correcting Implementation for Measuring Performance Counters

The biggest mistake I currently have is that my test applications are only measuring the counters of the master thread. This can be seen by the drop off of clock cycles counted when the number of threads increases. This number should remain relatively the same when counted between all threads, but from the perspective of the master thread it decreases as the work is divided up and given to other threads. Since my current method is fundamentally incorrect, this will be my first focus for the Winter quarter.

4.2 Change Method for Measuring Execution Time

Currently I have separate test applications for measuring execution time and performance counters, as I was concerned the code for measuring one would affect the measurement of the other. The main problem with this is I have to execute the procedures twice to get all the data, once for the execution time and once for the performance counters. Since the time and performance counter data are gathered on different executions of the test application, analyzing any relationship between the two would be inaccurate. I will look to correct this mistake going forward so that no outside factors that change between executions can affect future analysis.

4.3 Expanding Performance Counters and Applying Machine Learning

Once these mistakes are corrected for, I can look into gathering data on many more of the performance counters available on the WAVE HPC. With the introduction of these additional data points, visual analysis will become ineffective. So going forward I will begin using machine learning techniques to analyze the data in hopes of finding relationships between certain hardware events and the procedure execution time.

References

[1] W. da Silva Pereira, I. Kozachenko, J. Langou, J. Demmel, J. Dongarra, and J. Langou, *LAPACK—Linear Algebra PACKage*, 2022. LAPACK 3.11.0 is a software package provided by Univ. of Tennessee, Univ. of California, Berkeley, Univ. of Colorado Denver and NAG Ltd..

[2] Z. Xianyi, M. Kroeker, W. Saar, W. Qian, Z. Chothia, C. Shaohu, and L. Wen, *OpenBlas: An optimized BLAS library*, 2023. OpenBLAS 0.3.24.

[3] Intel Corporation, *Intel®-Optimized Math Library for Numerical Computing on CPUs & GPUs*, 2023. https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html.

[4] Santa Clara University, *WAVE High Performance Computing (HPC)*, 2023. https://wiki.wave.scu.edu/doku.php?id=wave_hpc.

[5] H. Jagode, A. Danalis, J. Dongarra, D. Barry, G. Congiu, and A. Pal, *PAPI: Performance Application Programming Interface*. The University of Tennessee, Knoxville, 2023. http://icl.utk.edu/papi/.