1. **Problem Statement**

Given the Matmul function in *matmul.c*, use OpenACC directives in the program to improve the performance. Due to GPU resources not available on Bridges, Stampede2 will be evaluated for testing OpenACC directives and performance using Knights Landing.

1. **Approach**

To best evaluate the difference in performance between the serial routine and the parallel routine, several changes had to be made to the source code to expose the program to the command line. These changes include allowing the matrix parameters (L, M, and N) to be defined and bound at runtime which requires the memory allocation for these matrices to be dynamic using malloc/calloc. To enable easier unit testing to ensure the program still calculated the cross product without errors, the debugger had to also be exposed to the command line. With these changes to the code, the Matmul program can be tested between source changes, specifically the OpenACC directives.

To streamline the testing process, the executive decision was made to create a Makefile to utilize the Make automation tool. This will not only speed up build times, building only when necessary, but also allow defining the parameters for each trial, names of executables, clean up routines, and test runs. Separate executables are to be built, as well, with one for the serial code without OpenACC flags and one with OpenACC compiler flags. Make run will verify these executables are up-to-date and run all the tests. All results are redirected to an output file named: results.txt. The current matmul function block will also be outputted to this file to maintain a record of the current edit for the results set generated.

Some logistical activities will also be required including inspection of Stampede2’s User Manual to be familiarized with the process for accessing resources, importing files, loading and utilizing local tools, and exporting results. Access to these systems will require updating the user accounts for both XSEDE and TACC with the additional requirement of linking to DUO’s two-step authentication. Importing and exporting files will require use of SCP or WinSCP for Windows users. The following outline lists the necessary steps towards completing this workflow.

**Workflow:**

1. Establish access to Texas Advanced Computing Center’s Stampede2 cluster
   1. Linking DUO account for XSEDE’s two-step authentication
   2. PuTTY into XSEDE’s SSO Hub: [username@log.xsede.org](mailto:username@log.xsede.org)
   3. Ssh to stampede2 through the SSO Hub: *gsissh stampede2*
2. Copy/paste matmul.c source code into a new document
3. Modify the source code for optimized testing
   1. Add OpenACC directives
   2. Expose matrix parameters (L, M, N) to the command line
   3. Expose debug mode to the command line
   4. Make memory allocation for matrices dynamic (since parameters are now determined at runtime)
4. Streamline unit testing with Makefile
   1. Define all trial parameters (TRIALS 1-10)
   2. Define executable names
   3. Define executable targets and corresponding dependencies and recipes for compiling the target
   4. Define clean routine
   5. Define run routine
      1. Check if targets need to be re-compiled
      2. Print the matmul function block
      3. Run serial runs
      4. Run OpenACC runs
   6. Define print routine (to keep track of runs)
      1. cat matmul.c source file
      2. grep the modified matmul function
      3. redirect to results.txt
   7. Define serial runs
      1. Echo “Serial Runs” redirect to results.txt
      2. Run trials 1-10 redirecting output to results.txt
   8. Define OpenACC runs
      1. Echo “Serial Runs” redirect to results.txt
      2. Run trials 1-10 redirecting output to results.txt
5. Build and Test
   1. Make the serial executable and OpenACC executable using Makefile
   2. Test OpenACC executable with debug flag to ensure changes have not broken the results matrix
   3. Run full test for trials 1-10 using both serial and OpenACC executable using Makefile
   4. Analyze results in results.txt file
   5. Modify the source code (OpenACC directives) for further testing
   6. Repeat until no further improvements can be made
6. Export all results to local machine for analysis and report
   1. Linking DUO account to TACC account’s two-step authentication
   2. Log in via WinSCP to copy files to local machine
7. **Solution**

During the process of implementing OpenACC to Matmul function, it was soon discovered that the current resources available do not allow for a comprehensive test of OpenACC capabilities. With Bridges GPU resources unavailable, Stampede2 was evaluated in an effort to utilize Knights Landing (Xeon Phi) x86 manycore processors. However, Stampede2 utilizes Knights Landing as a primary core rather than an accelerator card core. As such, the tests were unable to demonstrate the results had a GPU/accelerator been available.

However, this did not halt progress. With success in accessing Stampede2, the Matmul source file was successfully modified and compiled for testing. This required logging in and loading the GNU C compiler with version 7.1.0:

>> idev

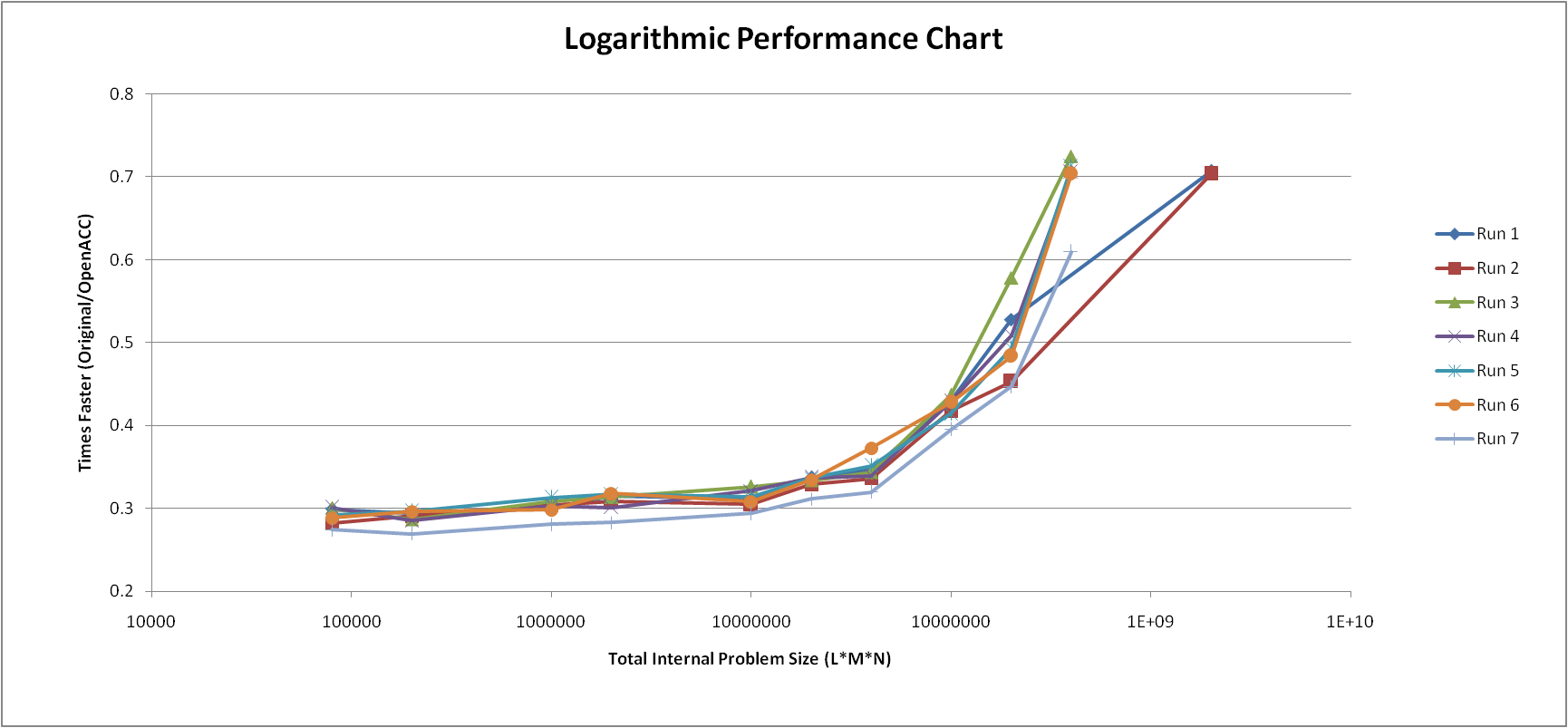
>> module load gcc/7.1.0

>> vim matmul.c

Once OpenACC directives were added to the source code, the executable generated depended on the flags used by the compiler. Without flags, the executable generated would be compiled ignoring OpenACC directives, but if OpenACC directives were to be included, the following command would be required:

>> gcc -fopenacc -fopenacc-dim=1024:1:128 -o <output file> <input source file>

The Makefile as outlined in the workflow above was also created to streamline this process and several tests were performed. Following the OpenACC guidelines, various directives and clauses were used in different tests. The follow results were generated from these trials as illustrated in Figure 1.



**Figure 1. The illustration above shows the results spanning seven different run and source code modifications (140 executions).**

A total of seven runs were performed, each with different modifications to the source code in terms of OpenACC directives. In each of these runs, ten separate trials were performed on two compiled executables: a serial executable and an executable utilizing OpenACC. This totaled 20 executed programs per run for a total of 140 executions to generate the chart shown above.

Various modifications were made from Run 1 to Run 7. These included runs with and without data regions, runs utilizing a copy clause versus those that utilized only copyin and copyout strategically, etc. Particular attention was paid to data movement to minimize the overhead incurred when copying in/out data. The reduction clause was also leveraged to ensure the summation step of the innermost loop caused no race conditions and evaluated correctly. The copyin data clause was utilized only for matrix A and B, since matrix C would be overwritten and would not require initialization. Matrix C was the only matrix designated for the copyout data clause, since matrix A and B were no longer needed. Each loop was also designated to be divided among the gangs, workers, and vectors available, with the inner-most loop designated to the vectors since these operations are SIMD. The optimized solution (Run 3) can be found illustrated below in Figure 2.

**int matmul( int l, int m, int n, float \*A, float \*B, float \* restrict C )**

**{**

int i, j, k;

**#pragma acc data copyin(A, B) copyout(C)**

{

**#pragma acc kernels loop gang**

for( i=0; i<l; i++ ) // Loop over the rows of A and C.

{

**#pragma acc worker**

for( k=0; k<n; k++ ) // Loop over the columns of B and C

{

int sum = 0;

**#pragma acc vector reduction (+:sum)**

// Initialize the output element for the inner

// product of row i of A with column j of B

for( j=0; j<m; j++ ) // Loop over the columns of A and C

{

sum += A[i\*m+j] \* B[j\*n+k]; // Compute the inner product

}

C[i\*n+k] = sum;

}

}

}

**}**

Figure 2. The Matmul function with optimized OpenACC directives is illustrated above. A data region is defined with specific copyin/copyout clauses. Reduction clause is also utilized.

Despite not having the resources to illustrate the actual performance improvements made with these changes, the tests were able to demonstrate the optimum solution for implementing OpenACC. Several additional tests were also performed using the previous OpenMP source code. These results can be found in the results.txt or illustrated in the Excel file attached.