# Programming Assignment #3

## 1. Hardware

The CSL machine was used for compilation, execution, and benchmarking.

- Processor model: Intel® Core™ i5-11500
- Number of cores: 6
- Number of threads: 12
- 50 GB/s max memory bandwidth

## 2. Operating System and Compilation:

The operating system was Ubuntu 22.04.2 LTS

The intel compiler on the CSL machine was activated using the command:

```
source /s/intelcompilers-2019/bin/iccvars.sh intel64
```

This setup MKL as needed.

Commands used for compilation and execution:

```
icc *.cpp -mkl -qopenmp -o main
./main
```

## 3. Code Changes

All the kernels used in the Conjugate Gradients algorithm were replaced with their MKL counterparts.

Timing Kernels:

An individual timer (Restart and Pause) was placed around the function call of each kernel. A timer was also placed around the Conjugate gradient function call to the time the entire execution.

Modifying kernels:

A separate implementation was created using the MKL kernels which would would execute if the variable DO_NOT_USE_MKL was not defined.

Saxpy Kernels: (z = x * scale + y)

There were three versions of this kernel implemented.

The first one was used when none of the arguments passed into the kernel were aliases of each other. This was only called once in the conjugate gradients algorithm.

```
// Version 1
void Saxpy(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM][ZDIM],
    float (&z)[XDIM][YDIM][ZDIM],
    const float scale)
{
#ifdef DO_NOT_USE_MKL
#pragma omp parallel for
    for (int i = 0; i < XDIM; i++)
    for (int j = 0; j < YDIM; j++)
    for (int k = 0; k < ZDIM; k++)
        z[i][j][k] = x[i][j][k] * scale + y[i][j][k];
#else
    // z = y
    cblas_scopy(
        XDIM * YDIM * ZDIM,
        &y[0][0][0],
        1,
        &z[0][0][0],
        1
    );
    // z += scale * x
    cblas_saxpy(
        XDIM * YDIM * ZDIM,
        scale,
        &x[0][0][0],
        1,
        &z[0][0][0],
        1
    );
#endif
```

The second was used when z and y were aliases of each other and the function was effectively z += x * scale. This implementation was provided in the starter code for the assignment.

The third was when z and x were aliases of each other. This required the use of two mkl kernels cblas_sscal and cblas_saxpy. Due to this, this implementation was not very efficient and the non-mkl implementation was much faster.

```
// Version 3
void SaxpyV3(const float (&x)[XDIM][YDIM][ZDIM], float (&y)[XDIM][YDIM][ZDIM],
    const float scale)
{
    // y = scale * y + x
#ifdef DO_NOT_USE_MKL
    Saxpy(y,x,y,scale);
#else
    // y = scale * y
    cblas_sscal(
        XDIM * YDIM * ZDIM,
        scale,
        &y[0][0][0],
        1
    );
    // y += 1.0 * x
    cblas_saxpy(
        XDIM * YDIM * ZDIM,
        1.0f,
        &x[0][0][0],
        1,
        &y[0][0][0],
        1
    );
#endif
}
```

# 4. Performance Figures

Implementation of kernels using OpenMP:

```
[nikunj@rockhopper-01] (21)$ ./main
[Initialization : 5850.41ms]
Conjugate Gradients terminated after 256 iterations; residual norm (nu) = 0.00097589
[Total Laplacian Time : 19469ms]
[Total InnerProduct Time : 3652.08ms]
[Total Norm Time : 852.736ms]
[Total Copy Time : 2805.5ms]
[Total Saxpy Version 1 Time : 11.1485ms]
[Total Saxpy Version 2 Time : 5716.53ms]
[Total Saxpy Version 3 Time : 2859.47ms]
[Total Conjugate Gradients Time : 35366.9ms]
```

Implementation of kernels using MKL:

```
[nikunj@rockhopper-01] (24)$ ./main
[Initialization : 5847.09ms]
Conjugate Gradients terminated after 256 iterations; residual norm (nu) = 0.00097589
[Total Laplacian Time : 17637.7ms]
[Total InnerProduct Time : 3144.89ms]
[Total Norm Time : 821.273ms]
[Total Copy Time : 1722.92ms]
[Total Saxpy Version 1 Time : 18.5676ms]
[Total Saxpy Version 2 Time : 5355.98ms]
[Total Saxpy Version 3 Time : 4450.66ms]
[Total Conjugate Gradients Time : 33152.8ms]
```

# 5. Comments

As it can be seen from the performance figures above, an acceleration was seen in most of the implementations when using MKL kernels. The Laplacian, InnerProduct, Norm, Copy and Saxpy Version 2 all experienced better performance.

The saxpy version 1 and 3 kernels suffered a loss in performance because more than one MKL kernel needed to be used to achieve the intended effects of the functions.

However, the overall performance of the conjugate gradients algorithm still improved and if a mixture of mkl and our own implementations were used, the performance would be significantly better. The residual norm values and the number of iterations taken by the algorithm remained consistent across implementations.