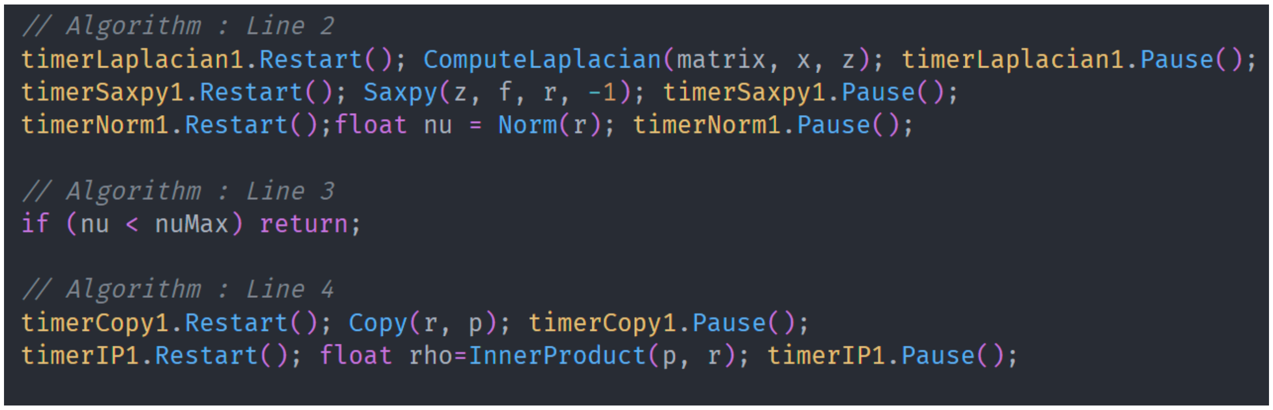
CS639: Homework 3 (git) Rajesh Shashi Kumar

1 Machine Configuration

Attribute	**Value**
Hostname	barolo.cs.wisc.edu
OS	Ubuntu 20.04.5 LTS
Compiler	g++ (gcc version 9.4.0), OpenMP version 4.5
Compile command	*(Makefile included in root directory of zip file)* *icc *.cpp -Wall -O3 -o conjugate_gradients_mkl -qopenmp -mkl*
CPU	AMD EPYC 7451 24-Core Processor
Cache configuration	L1-3MiB, L2-24MiB, L3-128MiB
Memory bandwidth	~150 GiB/s (Source,2). The machine has 256GB of memory spread across (8 out of 16) slots each housing 32GB stick. This information was retrieved using *sudo lshw -class memory*
Number of threads	24 cores * 2 threads/core = 48 threads
Dependencies	GCC, OpenMP, ICC (source /s/intelcompilers-2019/bin/iccvars.sh intel64)

Changes to the base code (LaplaceSolver_1_5)

- Added MKL variants for InnerProduct, Norm and Copy kernels
- Instrumented timers for each invocation for every kernel

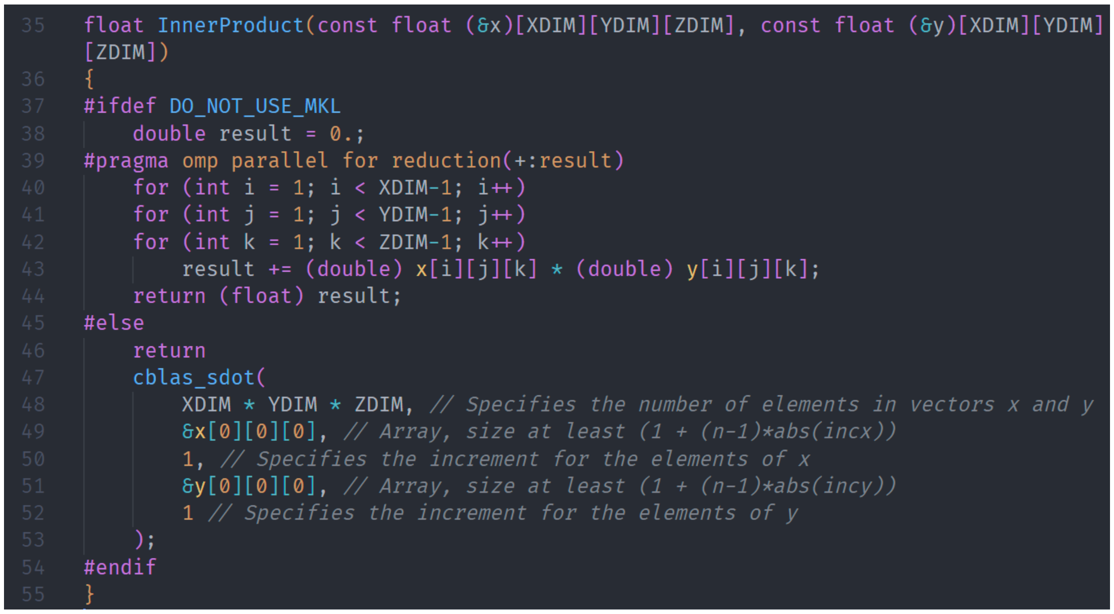


```
// Algorithm : Line 2
timerLaplacian1.Restart(); ComputeLaplacian(matrix, x, z); timerLaplacian1.Pause();
timerSaxpy1.Restart(); Saxpy(z, f, r, -1); timerSaxpy1.Pause();
timerNorm1.Restart();float nu = Norm(r); timerNorm1.Pause();

// Algorithm : Line 3
if (nu < nuMax) return;

// Algorithm : Line 4
timerCopy1.Restart(); Copy(r, p); timerCopy1.Pause();
timerIP1.Restart(); float rho=InnerProduct(p, r); timerIP1.Pause();
```

Fig 1: Excerpt to illustrate the instrumented timers for per-kernel measurements

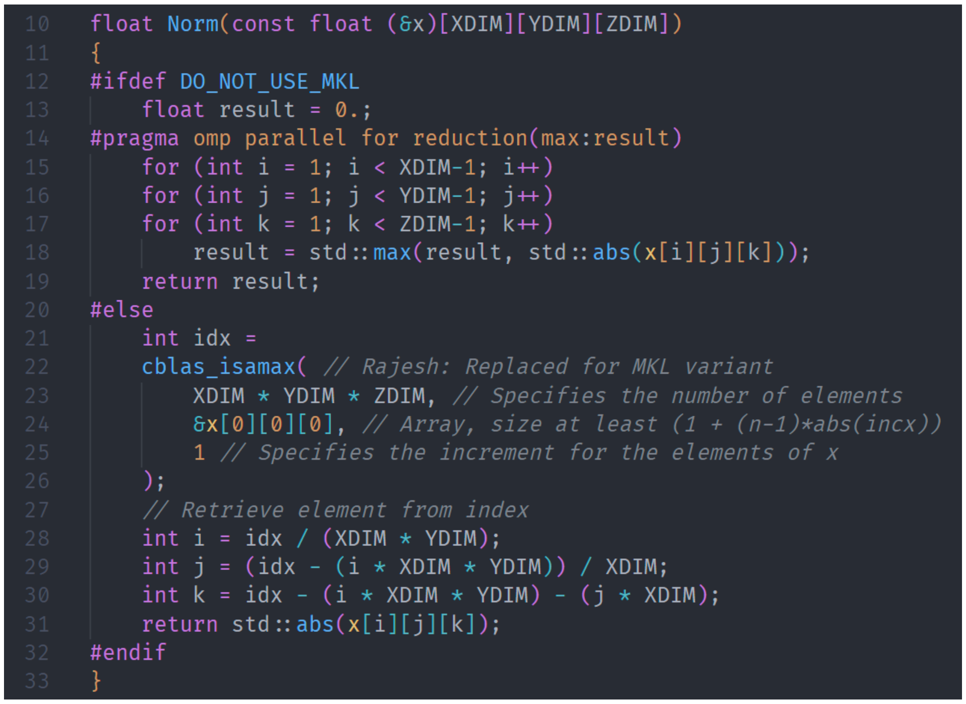
2 MKL variant replacement for Conjugate Gradient

I added the MKL variants for InnerProduct, Norm and Copy kernels in the Conjugate Gradient algorithm as shown in the code snippets below:



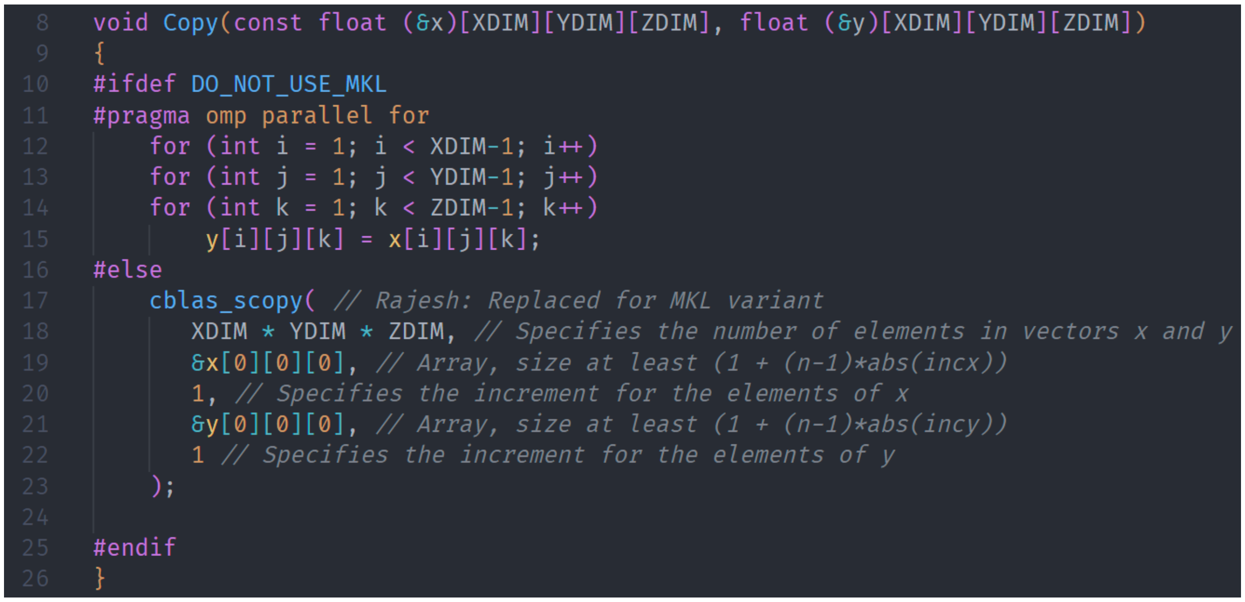
```
float InnerProduct(const float (&x)[XDIM][YDIM][ZDIM], const float (&y)[XDIM][YDIM]
[ZDIM])
{
#ifdef DO_NOT_USE_MKL
    double result = 0.;
#pragma omp parallel for reduction(+:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result += (double) x[i][j][k] * (double) y[i][j][k];
    return (float) result;
#else
    return
    cblas_sdot(
        XDIM * YDIM * ZDIM, // Specifies the number of elements in vectors x and y
        &x[0][0][0], // Array, size at least (1 + (n-1)*abs(incx))
        1, // Specifies the increment for the elements of x
        &y[0][0][0], // Array, size at least (1 + (n-1)*abs(incy))
        1 // Specifies the increment for the elements of y
    );
#endif
}
```

Fig 2: Inner product MKL variant using *cblas_sdot*



```
float Norm(const float (&x)[XDIM][YDIM][ZDIM])
{
#ifdef DO_NOT_USE_MKL
    float result = 0.;
#pragma omp parallel for reduction(max:result)
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        result = std::max(result, std::abs(x[i][j][k]));
    return result;
#else
    int idx =
    cblas_isamax( // Rajesh: Replaced for MKL variant
        XDIM * YDIM * ZDIM, // Specifies the number of elements
        &x[0][0][0], // Array, size at least (1 + (n-1)*abs(incx))
        1 // Specifies the increment for the elements of x
    );
    // Retrieve element from index
    int i = idx / (XDIM * YDIM);
    int j = (idx - (i * XDIM * YDIM)) / XDIM;
    int k = idx - (i * XDIM * YDIM) - (j * XDIM);
    return std::abs(x[i][j][k]);
#endif
}
```

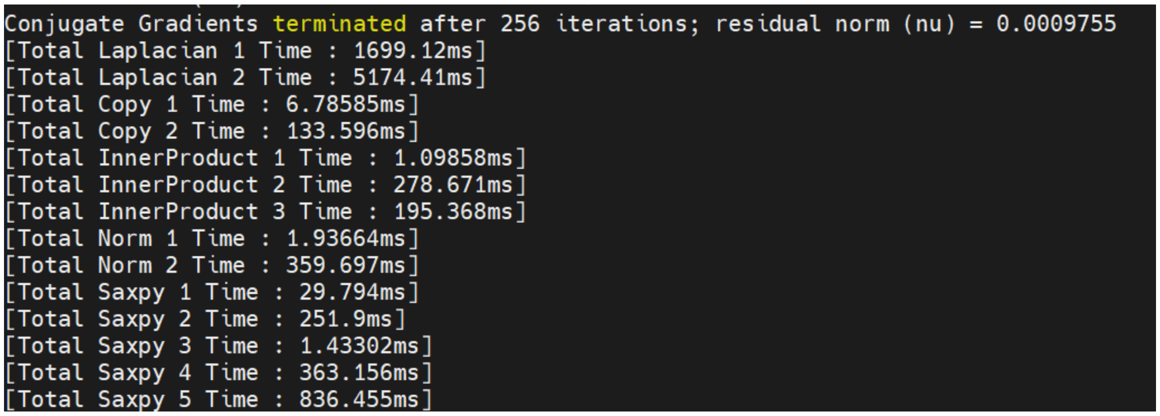
Fig 3: Norm MKL variant using *cblas_isamax*



```
void Copy(const float (&x)[XDIM][YDIM][ZDIM], float (&y)[XDIM][YDIM][ZDIM])
{
#ifdef DO_NOT_USE_MKL
#pragma omp parallel for
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++)
        y[i][j][k] = x[i][j][k];
#else
    cblas_scopy( // Rajesh: Replaced for MKL variant
       XDIM * YDIM * ZDIM, // Specifies the number of elements in vectors x and y
       &x[0][0][0], // Array, size at least (1 + (n-1)*abs(incx))
       1, // Specifies the increment for the elements of x
       &y[0][0][0], // Array, size at least (1 + (n-1)*abs(incy))
       1 // Specifies the increment for the elements of y
    );

#endif
}
```

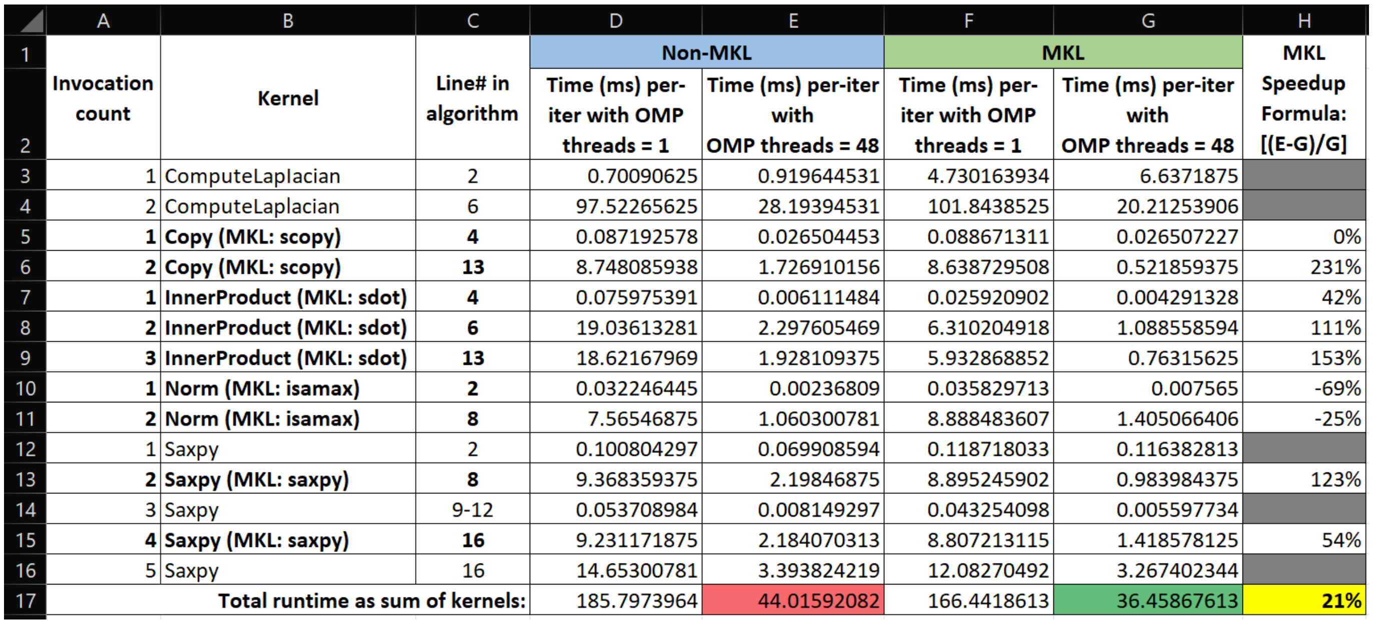
Fig 3: Copy MKL variant using *cblas_scopy*

The above changes were verified for functional correctness (in Fig 4) by ensuring consistency in the residual norm values. The performance results of these optimizations are presented in Fig 5.



```
Conjugate Gradients terminated after 256 iterations; residual norm (nu) = 0.0009755
[Total Laplacian 1 Time : 1699.12ms]
[Total Laplacian 2 Time : 5174.41ms]
[Total Copy 1 Time : 6.78585ms]
[Total Copy 2 Time : 133.596ms]
[Total InnerProduct 1 Time : 1.09858ms]
[Total InnerProduct 2 Time : 278.671ms]
[Total InnerProduct 3 Time : 195.368ms]
[Total Norm 1 Time : 1.93664ms]
[Total Norm 2 Time : 359.697ms]
[Total Saxpy 1 Time : 29.794ms]
[Total Saxpy 2 Time : 251.9ms]
[Total Saxpy 3 Time : 1.43302ms]
[Total Saxpy 4 Time : 363.156ms]
[Total Saxpy 5 Time : 836.455ms]
```

Fig 4: Functional verification upon MKL changes using OMP_THREADS=48



	A	B	C	D	E	F	G	H
1				**Non-MKL**		**MKL**		**MKL**
2	**Invocation count**	**Kernel**	**Line# in algorithm**	**Time (ms) per-iter with OMP threads = 1**	**Time (ms) per-iter with OMP threads = 48**	**Time (ms) per-iter with OMP threads = 1**	**Time (ms) per-iter with OMP threads = 48**	**Speedup Formula: [(E-G)/G]**
3	1	ComputeLaplacian	2	0.70090625	0.919644531	4.730163934	6.6371875	
4	2	ComputeLaplacian	6	97.52265625	28.19394531	101.8438525	20.21253906	
5	**1**	**Copy (MKL: scopy)**	**4**	0.087192578	0.026504453	0.088671311	0.026507227	0%
6	**2**	**Copy (MKL: scopy)**	**13**	8.748085938	1.726910156	8.638729508	0.521859375	231%
7	**1**	**InnerProduct (MKL: sdot)**	**4**	0.075975391	0.006111484	0.025920902	0.004291328	42%
8	**2**	**InnerProduct (MKL: sdot)**	**6**	19.03613281	2.297605469	6.310204918	1.088558594	111%
9	**3**	**InnerProduct (MKL: sdot)**	**13**	18.62167969	1.928109375	5.932868852	0.76315625	153%
10	**1**	**Norm (MKL: isamax)**	**2**	0.032246445	0.00236809	0.035829713	0.007565	-69%
11	**2**	**Norm (MKL: isamax)**	**8**	7.56546875	1.060300781	8.888483607	1.405066406	-25%
12	1	Saxpy	2	0.100804297	0.069908594	0.118718033	0.116382813	
13	**2**	**Saxpy (MKL: saxpy)**	**8**	9.368359375	2.19846875	8.895245902	0.983984375	123%
14	3	Saxpy	9-12	0.053708984	0.008149297	0.043254098	0.005597734	
15	**4**	**Saxpy (MKL: saxpy)**	**16**	9.231171875	2.184070313	8.807213115	1.418578125	54%
16	5	Saxpy	16	14.65300781	3.393824219	12.08270492	3.267402344	
17		**Total runtime as sum of kernels:**		185.7973964	44.01592082	166.4418613	36.45867613	**21%**

Fig 5: Performance results from using MKL optimizations for Conjugate Gradient

The MKL variants of the kernels that were replaced are indicated in bold on "column B" in Fig 5. The bottomline of the above results is that we see an overall speed-up in execution time of about 21%. Individually, the MKL counterparts of the kernels consistently outperforms baseline except in the case of Norm. This can likely be attributed to the fact that *cblas_i?amax* is more generic in function than we need for our purposes in the algorithm. The variations in MKL performance between independent invocations of the same kernel (say InnerProduct) can be attributed to three factors (i) statistical variation due to measurements (ii) background processes running on the system (iii) variations in the input data for each of the invocations.