Provided are 3 different parallel Matrix Multiply $C = A \times B$ implementations:
- A classic Matrix Multiply implementation
- A hand-optimized Matrix Multiply implementation
- A Matrix Multiply implementation that uses the ACML library

Parallelization of the Matrix Multiply:
- The Master process (rank 0) is responsible for scattering and gathering data
- We have $(n-1)$ workers; where $n$ is the # of processes
- The Master process divides Matrix A into $(n-1)$ sub-matrices and sends one sub-array to each worker
- Each worker performs the Matrix Multiply $C_{SUB} = A_{SUB} \times B$ and when finished, sends the result sub-array to the Master process
- The result of the two manual Matrix Multiply implementations is verified with the dgemm() BLAS call.

Usage of the programs:
$ aprun -n 2 ./program-name N seed$
- You have to run the programs with $n \geq 2$ processes
- $N$ (int) specifies the matrix size of one dimension
e.g. if $N=100$: $C[100][100] = A[100][100] \times B[100][100]$
- $seed$ (int $\geq 0$) is the parameter to generate random values for Matrix A and B. The same seed generates the exact same random numbers. This might be helpful in case you need to debug or/and check the correctness of the result matrix. Different seed values generate different random numbers.

Your job is to instrument, analyze and compare the performance of the 3 Matrix Multiply implementations using PAPI.

1) Instrumentation:
- Use the low-level API: it gives you the advantage that you can measure native events as well as preset events. i.e. you are not restricted to the preset events only as is the case for the high-level API.
- I added TODO comments in the code to guide you where to place the appropriate PAPI calls.
- Make sure you check the return values of those calls.
2) Analysis:

- Collect hardware performance events to measure, e.g.:
  - Number of floating point operations (verify those numbers with the theoretical number of operations)
  - Number of L1/L2 data cache misses
- Collect data for different matrix sizes, compare those values and **draw conclusions**
- **Compare the collected hardware event values for all three code versions.** Are the values different for different implementations (especially the cache access, miss, hit values)? If so, make sure you understand and explain why! Also take the size of the L1 and L2 caches into account. This information for Kraken is given in the lecture material.

**NOTE:** when analyzing the performance of an application, one usually ends up with tons of data. Comparing the data and organizing the comparisons in order to get useful insight into the performance of an application is challenging but important.

**General:**

1. Log into Kraken:
   
   % ssh username@kraken.nics.tennessee.edu
   
   for information on Kraken:
   
   [http://www.nics.tennessee.edu/computing-resources/kraken](http://www.nics.tennessee.edu/computing-resources/kraken)

2. Copy /nics/c/home/hjagode/public/cs594 to your /lustre/scratch/username directory

3. Load PAPI and ACML module: % module load xt-papi acml

4. Compile the NULL solution: % make clean && make

5. Submit jobs using the provided batch script go.sh: % qsub go.sh