```
// ------------------------------------------------------------------------
// matrix multiplication
//
// 0. choose appropriate compiler version e.g.
//
//       module load intelcompiler
//
// 1. compile with openmp support enabled and MKL library included
//
//       icc -O3 -openmp -openmp-report2 -lmkl matmult.c -o matmult
//
// 2. set number of threads and prohibit dynamic adjustment of number
//
//       export set OMP_DYNAMIC=FALSE
//       export set OMP_NUM_THREADS=4
//
//    optionally bind threads to processes by *one* of the following:
//
//       export set GOMP_CPU_AFFINITY="0-7:2"
//
//       export set KMP_AFFINITY=
//         "verbose,granularity=core,explicit,proclist=[0,1,2,3]"
//
//    optionally apply runtime scheduling ("schedule(runtime)")
//
//       export set OMP_SCHEDULE="static,1"
//
// 3. prepare runtime monitoring in other window showng all threads
//
//      top -u <username> -H
//
// 4. execute with timing switched on
//
//      time ./matmult
//
// ------------------------------------------------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

#include "mkl_gmp.h"

#define N 400

mpz_t A[N][N], B[N][N], C[N][N];

int main(int argc, char *argv[])
{
  int i, j, k;
  mpz_t s, p;

  // initialize A and B
  for (i=0; i<N; i++)
  {
    for (j=0; j<N; j++)
    {
      mpz_init(A[i][j]);
      mpz_init(B[i][j]);
      mpz_init(C[i][j]);

      mpz_set_si(A[i][j], rand()); // A[i][j] = rand()
      mpz_set_si(B[i][j], rand()); // B[i][j] = rand()
    }
  }
```

```
  // print part of A and B
  mpz_out_str(stdout, 10, A[0][0]); printf("\n");
  mpz_out_str(stdout, 10, B[0][0]); printf("\n");

  // ----------------------------------------------------------------
  // parallel matrix multiplication:
  // iterations of outer loop are distributed among multiple threads
  // 1. make sure that local variables of outer loop are thread-private
  // 2. choose runtime schedule (environment variable OMP_SCHEDULE)
  // ----------------------------------------------------------------
  #pragma omp parallel for private(j,k,s,p) schedule(runtime)
  for (i=0; i<N; i++)
  {
    // s and p must be initialized within each thread
    mpz_init(s);
    mpz_init(p);
    for (j=0; j<N; j++)
    {
      mpz_set_si(s, 0);                // s = 0
      for (k=0; k<N; k++)
      {
        mpz_mul(p, A[i][k], B[k][j]); // p = A[i][k]*B[k][j]
        mpz_add(s, s, p);             // s = s+p
      }
      mpz_set(C[i][j], s);            // C[i][j] = s
    }
  }

  // print part of C
  mpz_out_str(stdout, 10, C[0][0]); printf("\n");
}
```